



# **DBKDA 2014**

The Sixth International Conference on Advances in Databases, Knowledge, and  
Data Applications

ISBN: 978-1-61208-334-6

April 20 - 24, 2014

Chamonix, France

## **DBKDA 2014 Editors**

Friedrich Laux, Reutlingen University, Germany

Andreas Schmidt, Karlsruhe University of Applied Sciences | Karlsruhe Institute of  
Technology, Germany

Kiyoshi Nitta, Yahoo Japan Research, Japan

Iztok Savnik, Jozef Stefan Institute and University of Primorska, Slovenia

# DBKDA 2014

## Foreword

The Sixth International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA 2014), held between April 20 - 24, 2014 in Chamonix, France, continued a series of international events covering a large spectrum of topics related to advances in fundamentals on databases, evolution of relation between databases and other domains, data base technologies and content processing, as well as specifics in applications domains databases.

Advances in different technologies and domains related to databases triggered substantial improvements for content processing, information indexing, and data, process and knowledge mining. The push came from Web services, artificial intelligence, and agent technologies, as well as from the generalization of the XML adoption.

High-speed communications and computations, large storage capacities, and load-balancing for distributed databases access allow new approaches for content processing with incomplete patterns, advanced ranking algorithms and advanced indexing methods.

Evolution on e-business, ehealth and telemedicine, bioinformatics, finance and marketing, geographical positioning systems put pressure on database communities to push the 'de facto' methods to support new requirements in terms of scalability, privacy, performance, indexing, and heterogeneity of both content and technology.

DBKDA 2014 also featured the following Workshop:

- GraphSM 2014: The First International Workshop on Large-scale Graph Storage and Management

We take here the opportunity to warmly thank all the members of the DBKDA 2014 Technical Program Committee, as well as the numerous reviewers. The creation of such a high quality conference program would not have been possible without their involvement. We also kindly thank all the authors who dedicated much of their time and efforts to contribute to DBKDA 2014. We truly believe that, thanks to all these efforts, the final conference program consisted of top quality contributions.

Also, this event could not have been a reality without the support of many individuals, organizations, and sponsors. We are grateful to the members of the DBKDA 2014 organizing committee for their help in handling the logistics and for their work to make this professional meeting a success.

We hope that DBKDA 2014 was a successful international forum for the exchange of ideas and results between academia and industry and for the promotion of progress in the fields of databases, knowledge and data applications.

We are convinced that the participants found the event useful and communications very open. We also hope the attendees enjoyed the charm of Chamonix, France.

**DBKDA 2014 Chairs:**

Friedrich Laux, Reutlingen University, Germany

Aris M. Ouksel, The University of Illinois at Chicago, USA

Serge Miranda, Université de Nice Sophia Antipolis, France

Andreas Schmidt, Karlsruhe University of Applied Sciences | Karlsruhe Institute of Technology, Germany

Maribel Yasmina Santos, University of Minho, Portugal

Filip Zavoral, Charles University Prague, Czech Republic

Maria Del Pilar Angeles, Universidad Nacional Autonoma de Mexico - Del Coyoacan, Mexico

# DBKDA 2014

## Committee

### DBKDA Advisory Chairs

Friedrich Laux, Reutlingen University, Germany  
Aris M. Ouksel, The University of Illinois at Chicago, USA  
Serge Miranda, Université de Nice Sophia Antipolis, France  
Andreas Schmidt, Karlsruhe University of Applied Sciences | Karlsruhe Institute of Technology, Germany  
Maribel Yasmina Santos, University of Minho, Portugal  
Filip Zavoral, Charles University Prague, Czech Republic  
Maria Del Pilar Angeles, Universidad Nacional Autonoma de Mexico - Del Coyoacan, Mexico

### DBKDA 2014 Technical Program Committee

Nipun Agarwal, Oracle Corporation, USA  
Reza Akbarinia, INRIA, France  
Suad Alagic, University of Southern Maine, USA  
Fabrizio Angiulli, University of Calabria, Italy  
Annalisa Appice, Università degli Studi di Bari, Italy  
Zeyar Aung, Masdar Institute of Science and Technology - Abu Dhabi, United Arab Emirates  
Fadila Bentayeb, University of Lyon 2, France  
Martine Cadot, LORIA-Nancy, France  
Michelangelo Ceci, University of Bari, Italy  
Chin-Chen Chang, Feng Chia University Taiwan, Taiwan  
Chi-Hua Chen, National Chiao Tung University - Taiwan, R.O.C.  
Qiming Chen, HP Labs - Palo Alto, USA  
Ding-Yuan Cheng, National Chiao Tung University, Taiwan , R.O.C.  
Camelia Constantin, UPMC, France  
Maria Del Pilar Angeles, Universidad Nacional Autonoma de Mexico - Del Coyoacan, Mexico  
Taoufiq Dkaki, IRIT - Toulouse, France  
Cédric du Mouza, CNAM - Paris, France  
Gledson Elias, Universidade Federal da Paraiba, Brazil  
Bijan Fadaeenia, Islamic Azad University- Hamedan Branch, Iran  
Victor Felea, "A. I. Cuza" University of Iasi, Romania  
Ingrid Fischer, University of Konstanz, Germany  
Eloy Gonzales, National Institute of Information and Communications Technology - Kyoto, Japan  
Robert Gottstein, Technische Universität Darmstadt, Germany  
Martin Grund, Hasso-Plattner-Institute - Potsdam, Germany  
Ismail Hababeh, United Arab Emirates University - Al-Ain, UAE  
Phan Nhat Hai, Lirmm Lab - University Montpellier 2, France  
Takahiro Hara, Osaka University, Japan  
Bingsheng He, Nanyang Technological University, Singapore  
Tobias Hoppe, Ruhr-University of Bochum, Germany  
Wen-Chi Hou, Southern Illinois University, USA  
Edward Hung, The Hong Kong Polytechnic University - Hong Kong, PRC

Dino Ienco, Irstea Montpellier, France  
Chris Ireland, Independent Scientist, UK  
Yasunori Ishihara, Osaka University, Japan  
Vladimir Ivancevic, University of Novi Sad, Serbia  
Savnik Iztok, University of Primorska, Slovenia  
Wassim Jaziri, ISIM Sfax, Tunisia  
Sungwon Jung, Sogang University - Seoul, Korea  
Mehdi Kargar, York University, Toronto, Canada  
Nhien An Le Khac, University College Dublin, Ireland  
Sadegh Kharazmi, RMIT University - Melbourne, Australia  
Kyoung-Sook Kim, National Institute of Information and Communications Technology, Japan  
Daniel Kimmig, Karlsruhe Institute of Technology, Germany  
Christian Kop, University of Klagenfurt, Austria  
Zineddine Kouahla, University of Nantes, France  
Jens Krueger, Hasso Plattner Institute / University of Potsdam, Germany  
Friedrich Laux, Reutlingen University, Germany  
Christophe Leblay, University of Jyväskylä, Finland  
Alain Lelu, University of Franche-Comté, France  
Carson Leung, University of Manitoba, Canada  
Sebastian Link, The University of Auckland, New Zealand  
Chunmei Liu, Howard University, USA  
Corrado Loglisci, University of Bari, Italy  
Qiang Ma, Kyoto University, Japan  
Murali Mani, University of Michigan - Flint, USA  
Gerasimos Marketos, University of Piraeus, Greece  
Ernestina Menasalvas, Universidad Politécnica de Madrid, Spain  
Elisabeth Métais, CEDRIC / CNAM - Paris, France  
Cristian Mihaescu, University of Craiova, Romania  
Serge Miranda, Université de Nice Sophia Antipolis, France  
Mehran Misaghi, Educational Society of Santa Catarina – Joinville, Brazil  
Mohamed Mkaouar, Sfax, Tunisia  
Jacky Montmain, LGI2P - Ecole des Mines d'Alès, France  
Yasuhiko Morimoto, Hiroshima University, Japan  
Bela Mutschler, Ravensburg-Weingarten University of Applied Sciences, Germany  
Franco Maria Nardini, ISTI-CNR, Italy  
Khaled Nagi, Alexandria University, Egypt  
Aris M. Ouksel, The University of Illinois at Chicago, USA  
George Papastefanatos, Athena Research and Innovation Center, Greece  
Alexander Pastwa, Ruhr-Universität Bochum, Germany  
Dhaval Patel, IIT-Roorkee, Singapore  
Przemyslaw Pawluk, York University - Toronto, Canada  
Alexander Peter, AOL Data Warehouse, USA  
Alain Pirotte, University of Louvain (Louvain-la-Neuve), Belgium  
Pascal Poncelet, LIRMM, France  
Mandar Rahurkar, Yahoo! Labs, USA  
Praveen R. Rao, University of Missouri-Kansas City, USA  
Mathieu Roche, TETIS, Cirad, France  
Satya Sahoo, Case Western Reserve University, USA

Fatiha Saï's, LRI (CNRS & Paris Sud University), France  
Abhishek Sanwaliya, Indian Institute of Technology - Kanpur, India  
Ismael Sanz, Universitat Jaume I - Castelló, Spain  
M. Saravanan, Ericsson India Pvt. Ltd -Tamil Nadu, India  
Idrissa Sarr, Université Cheikh Anta Diop, Senegal  
Najla Sassi Jaziri, ISSAT Mahdia, Tunisia  
Andreas Schmidt, Karlsruhe University of Applied Sciences | Karlsruhe Institute of Technology, Germany  
Yong Shi, Kennesaw State University, USA  
Damires Souza, Federal Institute of Education, Science and Technology of Paraiba (IFPB), Brazil  
Lubomir Stanchev, University-Purdue University - Fort Wayne, USA  
Ahmad Taleb, Najran University, Saudi Arabia  
Maguelonne Teisseire, Irstea - UMR TETIS, France  
Martin Theobald, Max Planck Institute for Informatics, Germany  
Kathryn Thornton, Brunel University, UK  
Gabriele Tolomei, CNR, Italy  
Jose Torres-Jimenez, CINVESTAV 3C, Mexico  
Nicolas Travers, CNAM-Paris, France  
Thomas Triplet, Wajam, Montreal, Canada  
Marian Vajtersic, University of Salzburg, Austria  
Genoveva Vargas, Solar | CNRS | LIG-LAFMIA, France  
Fan Wang, Microsoft Corporation - Bellevue, USA  
Zhihui Wang, Dalian University of Technology, China  
Guandong Xu, Victoria University, Australia  
Maribel Yasmina Santos, University of Minho, Portugal  
Jin Soung Yoo, Indiana University-Purdue University - Fort Wayne, USA  
Filip Zavoral, Charles University Prague, Czech Republic  
Wei Zhang, Amazon.com, USA

### **GraphSM Chairs**

Kiyoshi Nitta, Yahoo Japan Research, Japan  
Iztok Savnik, Jozef Stefan Institute and University of Primorska, Slovenia

### **GraphSM 2014 Technical Program Committee**

Milan Djordjevic, University of Primorska, Slovenia  
Blaz Fortuna, Jozef Stefan Institute, Slovenia  
Dimitar Hristovski, University of Ljubljana, Slovenia  
Yasunori Ishihara, Osaka University, Japan  
Dejan Lavbic, University of Ljubljana, Slovenia  
Wolfgang May, University of Goettingen, Germany  
Khaled Nagi, Alexandria University, Egypt  
Kiyoshi Nitta, Yahoo Japan Research, Japan  
Iztok Savnik, Jozef Stefan Institute and University of Primorska, Slovenia  
Andreas Schmidt, Karlsruhe University of Applied Sciences & Karlsruhe Institute of Technology, Germany  
Tatjana Wezler, University of Maribor, Slovenia  
Kokou Yetongnon, Université de Bourgogne, France

## Copyright Information

For your reference, this is the text governing the copyright release for material published by IARIA.

The copyright release is a transfer of publication rights, which allows IARIA and its partners to drive the dissemination of the published material. This allows IARIA to give articles increased visibility via distribution, inclusion in libraries, and arrangements for submission to indexes.

I, the undersigned, declare that the article is original, and that I represent the authors of this article in the copyright release matters. If this work has been done as work-for-hire, I have obtained all necessary clearances to execute a copyright release. I hereby irrevocably transfer exclusive copyright for this material to IARIA. I give IARIA permission to reproduce the work in any media format such as, but not limited to, print, digital, or electronic. I give IARIA permission to distribute the materials without restriction to any institutions or individuals. I give IARIA permission to submit the work for inclusion in article repositories as IARIA sees fit.

I, the undersigned, declare that to the best of my knowledge, the article does not contain libelous or otherwise unlawful contents or invading the right of privacy or infringing on a proprietary right.

Following the copyright release, any circulated version of the article must bear the copyright notice and any header and footer information that IARIA applies to the published article.

IARIA grants royalty-free permission to the authors to disseminate the work, under the above provisions, for any academic, commercial, or industrial use. IARIA grants royalty-free permission to any individuals or institutions to make the article available electronically, online, or in print.

IARIA acknowledges that rights to any algorithm, process, procedure, apparatus, or articles of manufacture remain with the authors and their employers.

I, the undersigned, understand that IARIA will not be liable, in contract, tort (including, without limitation, negligence), pre-contract or other representations (other than fraudulent misrepresentations) or otherwise in connection with the publication of my work.

Exception to the above is made for work-for-hire performed while employed by the government. In that case, copyright to the material remains with the said government. The rightful owners (authors and government entity) grant unlimited and unrestricted permission to IARIA, IARIA's contractors, and IARIA's partners to further distribute the work.

## Table of Contents

Feature Construction for Time Ordered Data Sequences <i>Michael Schaidnagel and Fritz Laux</i>	1
Information Integration with Uncertainty: Performance <i>Inder Dumpa, Raja Kota, and Fereidoon Sadri</i>	7
Universal Evaluation System Data Quality <i>Maria del Pilar Angeles, Francisco Javier Garcia-Ugalde, Carlos Ortiz, Ricardo Valencia, Jhovany Pelcastre, Eduardo Reyes, and Arturo Nava</i>	13
A Semi-automatic Method to Fuzzy-Ontology Design by using Clustering and Formal Concept Analysis <i>Amira Aloui, Alaa Ayadi, and Amel Grissa-Touzi</i>	19
Modeling Ontology-based User Profiles from Company Knowledge <i>Silvia Calegari and Matteo Dominoni</i>	26
A Concept for Plagiarism Detection Based on Compressed Bitmaps <i>Andreas Schmidt, Reinhold Becker, Daniel Kimmig, Robert Senger, and Steffen Scholz</i>	30
Enriching Dimension Hierarchies with Topological Relations to Improve the Development of Spatial Data Warehouse <i>Sana Ezzedine, Sami Yassine Turki, and Sami Faiz</i>	35
In-Memory Distance Threshold Queries on Moving Object Trajectories <i>Michael Gowanlock and Henri Casanova</i>	41
Exploiting the Social Structure of Online Media to Face Transient Heavy Workload <i>Ibrahima Gueye, Idrissa Sarr, and Hubert Naacke</i>	51
Sample Trace: Deriving Fast Approximation for Repetitive Queries <i>Feng Yu, Wen-Chi Hou, and Cheng Luo</i>	59
Efficient Aggregate Cache Revalidation in an In-Memory Column Store <i>Stephan Muller, Lars Butzmann, and Hasso Plattner</i>	66
Enterprise Data Solution Leveraging Data Warehousing for Big Data Veracity at Saudi Aramco <i>Muhammad Khakwani</i>	74
Parallel In-Memory Distance Threshold Queries on Trajectory Databases <i>Michael Gowanlock, Henri Casanova, and David Schanzenbach</i>	80



A Database Synchronization Approach for 3D Simulation Systems <i>Martin Hoppen and Juergen Rossmann</i>	84
Achieving High Availability in D-Box <i>Miroslav Cermak and Filip Zavoral</i>	92
Trustworthy Laboratory Automation <i>Jan Potthoff, Dominic Lutjohann, and Nicole Jung</i>	98
Toward a New Approach of Distributed Databases Design and Implementation <i>Fadoua Hassen and Amel Grissa Touzi</i>	104
An Object-oriented Approach for Extending MySQL into NoSQL with Enhanced Performance and Scalability <i>Hyunju Shim, YoungChul Sohn, YoulWoong Sung, Yonggoo Kang, Iljoo Kim, and Ohoon Kwon</i>	111
Efficient Data Integrity Checking for Untrusted Database Systems <i>Anderson Luiz Silverio, Ricardo Felipe Custodio, Marcelo Carlomagno Carlos, and Ronaldo dos Santos Mello</i>	118
Quantifying the Elasticity of a Database Management System <i>Christian Tinnefeld, Daniel Taschik, and Hasso Plattner</i>	125
Hierarchical Piecewise Linear Approximation <i>Vineetha Bettaiah and Heggere Ranganath</i>	132
Cache Management for Aggregates in Columnar In-Memory Databases <i>Stephan Muller, Ralf Diestelkamper, and Hasso Plattner</i>	139
Survey of RDF Storage Managers <i>Kiyoshi Nitta and Iztok Sarnik</i>	148
Design of Distributed Storage Manager for Large-Scale RDF Graphs <i>Iztok Sarnik and Kiyoshi Nitta</i>	154
Learning Links in MeSH Co-occurrence Network: Preliminary Results <i>Andrej Kastrin and Dimitar Hristovski</i>	161

# Feature Construction for Time Ordered Data Sequences

Michael Schaidnagel  
 School of Computing  
 University of the West of Scotland  
 Email: B00260359@studentmail.uws.ac.uk

Fritz Laux  
 Faculty of Computer Science  
 Reutlingen University  
 Email: fritz.laux@reutlingen-university.de

**Abstract**—The recent years and especially the Internet have changed the way on how data is stored. We now often store data together with its creation time-stamp. These data sequences potentially enable us to track the change of data over time. This is quite interesting, especially in the e-commerce area, in which classification of a sequence of customer actions, is still a challenging task for data miners. However, before standard algorithms such as Decision Trees, Neuronal Nets, Naive Bayes or Bayesian Belief Networks can be applied on sequential data, preparations need to be done in order to capture the information stored within the sequences. Therefore, this work presents a systematic approach on how to reveal sequence patterns among data and how to construct powerful features out of the primitive sequence attributes. This is achieved by sequence aggregation and the incorporation of time dimension into the feature construction step. The proposed algorithm is described in detail and applied on a real life data set, which demonstrates the ability of the proposed algorithm to boost the classification performance of well known data mining algorithms for classification tasks.

**Index Terms**—feature construction, sequential data, temporal data mining

## I. INTRODUCTION

Huge amounts of data are being generated on a daily basis, in almost every aspect of our live. Advancements in computer science as well as computer hardware enable us to store and analyze these data. Especially in the e-commerce area it is common to log all user activities in an online shop. Such data can be ordered by their timestamp and can be allocated to data sequences of particular users. However, the logged activities or actions are not stored in a form that enables data mining right away. Therefore, it is important to preprocess the data before analyzing it (see also Han [1], Liu [2]). When data is only represented by primitive attributes and there is no prior domain expert knowledge available, the preprocessing task becomes challenging and creates the need for automated techniques. At this point attribute selection and/or feature construction techniques need to be applied. Attribute selection can be defined as the task of selecting a subset of attributes, which are able to perform at least as good on a given data mining task as the primitive (original) attributes set. The original values of the data set are called attributes, while the constructed data are called features. It is possible that primitive attributes are not able to adequately describe eventually existing relations among primitive attributes. Such interrelations (or also called interactions, see Shafti [3]) can

occur in a data set, if the relation between one attribute and the target concept depends on another attribute (see also Shafti [4]). Attribute selection alone can fail to find existing interaction among data. Therefore, one goal for feature construction is to find and highlight interactions. Feature construction can be defined as the process of creating new compound properties using functional expressions on the primitive attributes. Shafti [3] distinguishes between two types of features construction techniques in terms of their construction strategy:

- hypothesis-driven: create features based on a hypothesis (which is expressed as a set of rules). These features are then added to the original data set and are used for the next iteration in which a new hypothesis will be tested. This process continues until a stopping requirement is satisfied.
- data-driven methods: create features based on pre-determined functional expressions, which are applied on combinations of primitive features of a data set. These strategies are normally non-iterative and the new features are evaluated by directly assessing the data.

### A. Problem description

Both feature construction strategies are not able to include a dimension that is unique to sequential data: the time elapsed between the corresponding actions. The so far described strategies are not able to express a pattern, which occurs in the course of time. Reason for this is their focus on tuples (rows) in a database. In this work, we will focus on the data-driven strategy and propose a new technique that is able to find patterns that are spread across several rows of a sequence. This can be achieved by creating meaningful features that are able to transform sequence information into the tuple-space.

### B. Structure of the paper

The remainder of this work is structured as follows: Section II will give a short overview about the related literature. Subsection II-A will highlight our contribution to the particular research field. Our proposed algorithm takes additional consideration on sequential data. The characteristics of such data is described in Section III. Our approach to feature construction will be described in detail in Section IV. We divided the algorithm into four logical parts, which are respectively described in the subsections of Section IV. This is followed by an

experimental analysis in Section V, in which we demonstrate the ability of our proposed algorithm to boost classification performance on a real life data set. The paper is concluded by the sections Conclusion (Section VI) and Future Work (Section VII).

## II. RELATED WORK

Earlier work in the field of feature construction was done by Setiono and Liu [5]. They used a neuronal network to construct features in an automatic way for continuous and discrete data. Pagallo [6] proposed FRINGE, which builds a decision tree based on the primitive attributes to find suitable boolean combinations of attributes near the fringe of the tree. The newly constructed features are then added to the initial attributes set and the process is repeated until no new features are created. Zupan and Bohanec [7] used a neuronal net for attribute selection and applied the resulting feature set on the well known C4.5 [8] induction algorithm. Feature construction can also be used in conjunction with linguistic fuzzy rule models. García [9] et al. use previously defined functions over the input variables in order to test if the resulting combination returns more information about the classification than the single variables.

However, in order to deal with increasing complexity of their genetic algorithm in the empirical part, García only used three functions (SUM{ $X_i, X_j$ }, PRODUCT{ $X_i, X_j$ }, SUBSTRACT\_ABS{ $X_i, X_j$ }) to enlarge the feature space. Another approach to feature construction, which utilizes a genetic algorithm, is described by Alfred [11]. Although, his approach is not using different functions to create new combinations of features, it can create a big variety of features since it is not limited to binary combination. That means that it is able to combine more than two attributes at a time. The genetic algorithm selects thereby the crossover points for the feature sequences. Another mentionable contribution to the field of feature construction was done by Shafti [4]. She describes MFE3/GA, a method that uses a global search strategy (i.e., finding the optimal solution) to reduce the original data dimensions and find new non-algebraic representations of features. Her primary focus is to find interactions between the original attributes (such as the interaction of several cards in a poker game that form a certain hand). Sia [12] proposes a 'Fixed-Length Feature Construction with Substitution' method called FLFCWS. It constructs a set that consist of randomly combined feature subsets. This allows initial features to be used more than once for feature construction.

### A. Contribution

We propose an automated algorithm that is able to systematically construct and assess suitable new features out of data sequences for binary classification tasks. It thereby is also able to utilize the time dimension in a sequence of actions in order to access information, which can have a significant impact on the discriminatory power of features. Thereby, the algorithm transforms sequential data into tuple-based data in a way, that allows standard algorithm such as Neuronal Networks,

Bayesian Belief Network, Decision Trees or Naive Bayes to be applied on sequential data.

So far, feature construction techniques build new features 'horizontally' by combining columns of a data set. We also apply this techniques with a larger variety of mathematical operators. In addition to that we include the time elapsed between data points. Our approach is novel, since we try to 'vertically' go down the time axis of a sequence and create features by combining numeric values (or its probabilities in terms of string attributes) of the corresponding occurrences. The original values are aggregated during the feature construction process and this allows to store sequence based information on tuple level. As a result of that, the above mentioned standard algorithms can be applied (not all are able to handle sequenced data sets).

## III. GENERAL CHARACTERISTICS OF SEQUENTIAL DATA

This work often refers to the term sequential data. Thereby we understand data, that can be ordered by time and can be attributed to logical units (i.e., the sequence). A simple example for that are sessions in an online shop. Customers can view products and put them into their shopping basket. Every action can be represented in a data base as a row  $r$  with several attributes  $a_i \in E$ . Each row is provided with a timestamp  $t$ . A row can be associated to a logical unit  $sid$  (in our case the session id). There are  $n$  sequences  $sid_n$  in a data set  $E$ . Each sequence  $sid_n$  consist of at least one row  $r$ . The number of rows in a sequence equals to the length of a sequence  $ls$ , so that  $1 \leq ls \leq m$ . Table I, depicts the general schema of sequential data: It is important

TABLE I: Schema of sequence data

$r$	$t$	$sid$	$a_1$	$a_2$	$\dots$	$a_i$	$s_{label}$
$r_1$	$t_1$	$sid_1$	$a_{1_1}$	$a_{2_1}$	$\dots$	$a_{i_1}$	0
$r_2$	$t_2$	$sid_1$	$a_{1_2}$	$a_{2_2}$	$\dots$	$a_{i_2}$	0
$r_3$	$t_3$	$sid_1$	$a_{1_3}$	$a_{2_3}$	$\dots$	$a_{i_3}$	0
$r_4$	$t_4$	$sid_2$	$a_{1_4}$	$a_{2_4}$	$\dots$	$a_{i_4}$	1
$r_5$	$t_5$	$sid_2$	$a_{1_5}$	$a_{2_5}$	$\dots$	$a_{i_5}$	1
$r_6$	$t_6$	$sid_2$	$a_{1_6}$	$a_{2_6}$	$\dots$	$a_{i_6}$	1
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
$r_m$	$t_m$	$sid_n$	$a_{1_m}$	$a_{2_m}$	$\dots$	$a_{i_m}$	$\dots$

to differentiate between the number of rows (or tuples)  $m$  of a data set and the number of sequences  $n$ . Sequence  $sid_1$ , for example, has a length  $ls$  of three and contains a matrix such

$$\text{as } sid_1 = \begin{Bmatrix} a_{1_1} & a_{2_1} & \dots & a_{i_1} \\ a_{1_2} & a_{2_2} & \dots & a_{i_2} \\ a_{1_3} & a_{2_3} & \dots & a_{i_3} \end{Bmatrix}$$

In order to use our proposed method, which is described in detail in the following section, the user has also to mark the following columns on a data set:

- $t$ : timestamp column that is used for temporal based features. It is used to calculate the time elapsed between the collected data points of a sequence.
- $sid$ : sequence identifier column that is used for sequence aggregation. It identifies events/objects that can be logically associated to one entity

- $s_{label}$ : the proposed algorithm requires a binary column as target value. This is needed in order to automatically calculate the information gain of newly constructed features. Every sequence must only have one label, i.e., a customer in an online shop is either a returning customer or not (it can not be both at the same time).

During the feature construction process, we will create a feature table, which includes the  $s_{id}$ ,  $s_{label}$  and the created features  $f_p \in S$ . Please refer to Table II, for a schema of such a table. The data sequences are aggregated on a tuple-

TABLE II: Schema of feature table

$s_{id}$	$f_1$	$f_2$	...	$f_p$	$s_{label}$
$s_{id_1}$	$f_{1_1}$	$f_{2_1}$	...	$f_{p_1}$	0
$s_{id_2}$	$f_{1_2}$	$f_{2_2}$	...	$f_{p_2}$	1
...	...	...	...	...	...
$s_{id_n}$	$f_{1_n}$	$f_{2_n}$	...	$f_{p_n}$	...

based level. This enable the application of many standard classification algorithms.

#### IV. FEATURE CONSTRUCTION FOR DATA SEQUENCES

Our goal is to extend and search the initial problem space as much as possible. Problem space is thereby defined through the primitive (original) attributes  $E$ , which are used to solve a binary classification task. The accessible feature space expands, if more features are constructed. Albeit, this leads to an increase in search time, it brings a higher chance to find discriminatory features. In order to keep things as simple as possible, we describe the algorithm in four different subsections, each describing a certain sort of features creation technique. Please note that the initial attributes are, in a first step, categorized in string and numeric attributes. Reason for this is, that not all described functions are applicable on string values. Please note, that after each feature construction technique, we normalize the newly generated features with min-max normalization, depicted in (1). This provides an easy way to compare values that are on different numerical scales or different units of measure.

$$Normalized(e_i) = \frac{e_i - E_{min}}{E_{max} - E_{min}}, \text{ for } E_{max} > E_{min} \quad (1)$$

The first Subsection IV-A will show construction techniques for both string and numeric attributes. The second Subsection IV-B describes construction techniques for string-only attributes. After that we will focus in the third Subsection IV-C on numeric-only construction techniques. Subsection IV-D concludes this section by describing temporal based feature construction techniques.

##### A. Distinct occurrences based features

The general idea for this type of technique is to analyze if different occurrences per sequence allows to discriminate between the given labels. Basically, we aggregate all sequences  $s_{id_n}$  and count the distinct occurrences (so no duplicates are counted) for each given string as well as numeric attribute  $a_{i_n}$ . The constructed features  $f_p$  are then collected in  $S$ ,

**Input:**  $E$  // set of string and numeric attributes

$s_{label} \in \{0, 1\}$  // single value label indication

**Def:**  $a_i \in E$  // single attribute or a column in a data set

$s_{id} = (r_1, r_2, \dots, r_m)$  // sequences of rows  $r$

$S = \emptyset$  // set of constructed features

**for each**  $a_i \in E$  {

**for each**  $s_{id} \in E$  {

$f_p := (|\{a_{i_n}\}|, s_{id}, s_{label})$

$S := S \cup f_p$

}

}

**return**  $S$

Fig. 1: Pseudo-code feature construction based on distinct occurrences per label

together with its corresponding sequence identifier  $s_{id}$  and the corresponding session label  $s_{label}$ . Please note that the sequence identifier  $s_{id}$  is unique in  $S$  (as opposed to  $E$ ). The corresponding pseudo-code is depicted in Fig. 1.

In order to assess the quality of the new constructed feature  $f_i$ , we calculate the average of all aggregated values per label  $s_{label} \in \{0, 1\}$ . The difference between both averages is called split and is calculated as depicted in (4).

$$avg_0 = avg(\{f_p \in S | s_{label} = 0\}) \quad (2)$$

$$avg_1 = avg(\{f_p \in S | s_{label} = 1\}) \quad (3)$$

$$split_{f_i} = \frac{|avg_0 - avg_1|}{avg_0 + avg_1} \quad (4)$$

##### B. Concatenation based features

Purpose of this type of feature construction is to highlight simpler interactions among data. We systematically concatenate every string attribute in pairs of two and then again, count the distinct value-pairs per sequence identifier. Thereby interactions such as, if  $a_1$  AND  $a_2$  have low value-pair variety for label 0, but a high value-pair variety for label 1, are highlighted. Even for data sets with a high number of different occurrences, this kind of feature construction will highlight distinct occurrences between both labels. This procedure is only applicable on string attributes. This approach is similar to most common column combinations that is described widely in the literature (e.g., [4], [7], [11]). However, we once again use this technique on a different abstraction layer since we aggregate via the sequence identifier  $s_{id}$ . The corresponding pseudo-code is depicted in Fig. 2.

The algorithm copies the input attribute list  $E$  for looping purposes into a second variable  $E_2$ . Right after the second loop, it deletes the current attribute from copied list ( $E_2 - a_{2i}$ ). Reason for this is to avoid the same features to occur twice, due to symmetric properties. If, for example, we combine column  $a_i = X$  and  $a_j = Y$  of a data set, we will yield feature  $XY$ . This feature will have the same variability per sequence as the vice versa feature  $YX$ . The construction of such features can be avoided by deleting the current feature from the copied feature list  $E_2$ .

**Input:**  $E$  // set of primitive string attributes  
 $s_{label} \in \{0, 1\}$  // single value label indication  
**Def:**  $a_i \in E$  // single attribute or a column in a data set  
 $s_{id} = (r_1, r_2, \dots, r_m)$  // sequences of rows  $r$   
 $S = \emptyset$  // set of constructed features  
 $E_2 = E$  // copy of  $E$ , used for looping  
 $con()$  // concatenates two values

```

for each  $a_i \in E$  {
  //remove  $a_i$  to avoid vice versa features
   $E_2 := E_2 - \{a_i\}$ 
  for each  $a_j \in E_2$  {
    for each  $s_{id} \in E$  {
       $f_p = (|(con(a_i, a_j))|, s_{id}, s_{label})$ 
       $S = S \cup f_p$ 
    }
  }
}
return  $S$ 

```

Fig. 2: Pseudo-code feature construction based on concatenated string attributes

### C. Numeric operator based features

The basic idea of this feature construction technique is to combine two numeric attributes with basic arithmetic operators such as "+", "-", "\*", or "/". Garcia [9] and Pagallo [6] for instance are using similar techniques with fewer operators. In addition to the repeated use of arithmetic operators we, once again, use the sequence identifier attribute to aggregate the constructed features for each sequence. Lets put this into an example: attributes  $a_i$  and  $a_j$  are combined with the multiplication operator "\*" for a sequence  $s_{id_1}$ . The resulting

feature  $f = a_i * a_j$  exists in the sequence  $s_{id_1} = \begin{Bmatrix} a_{i_1} & a_{j_1} \\ a_{i_2} & a_{j_2} \\ a_{i_3} & a_{j_3} \end{Bmatrix}$

The sequence consists of three data points. In the aggregation phase, we sum up the multiplied attributes for all sequences  $\sum_{j=1}^3 f_{ij}$ . This process is repeated for all possible combinations of numeric attributes for all of the above mentioned mathematical operators. The full pseudo-code is depicted in Fig. 3. For these technique, we also avoid vice versa features as described in previous Subsection IV-B.

### D. Temporal axis based features

The general idea for this feature construction technique is to use the time axis, which is displayed in each sequence by the time indicator column  $t$ . This is applicable for both, numeric as well as string attributes. However, for string attributes, there needs to be some preparations done, which are explained further down in this subsection. We continue here to describe the process for numeric attributes. What the algorithm basically does, is to multiply the time interval (e.g., days, hours, minutes), between earliest data point and the current data point with the numeric value of corresponding attribute, which results in a weighting.

**Input:**  $E$  // set of primitive numeric attributes  
 $s_{label} \in \{0, 1\}$  // single value label indication  
**Def:**  $a_i \in E$  // single attribute or a column in a data set  
 $s_{id} = (r_1, r_2, \dots, r_m)$  // sequences of rows  $r$   
 $S = \emptyset$  // set of constructed features  
 $E_2 = E$  // copy of  $E$ , used for looping  
 $O$  // set of arithmetic operators  
 $ls$  // length of a sequence  $s_{id}$

```

for each  $a_i \in E$  {
  //remove  $a_i$  to avoid vice versa features
   $E_2 := E_2 - \{a_i\}$ 
  for each  $a_j \in E_2$  {
    for each  $o \in O$  {
      for each  $s_{id} \in E$  {
         $f_p = (\sum_{i=1}^{ls} (a_i o a_j), s_{id}, s_{label})$ 
         $S = S \cup f_p$ 
      }
    }
  }
}
return  $S$ 

```

Fig. 3: Pseudo-code feature construction based on numeric attributes

Table III, shows this for two example sequences. We have two attributes  $a_i$  and  $a_j$  for two sequences as well as the  $t$  column. In order to calculate the temporal based feature for attribute sequence  $s_{id} = 1$  in terms of attribute  $a_i$ , we first have to calculate the time between the earliest data point of  $s_{id} = 1$  and each of the 'current' data points. In Table III, this is depicted by the  $\Delta time\_in\_days$  column. The next step is to multiply the value of each  $t_i$  in  $s_{id} = 1$  with its corresponding delta time value:  $(a_{i_1} * 1, a_{i_2} * 10, \dots, a_{i_4} * 23)$ . The sum of this value is the new time based constructed feature  $f_p$ . This process is repeated for all sequences  $s$  and for all numerical attributes  $E$ .

TABLE III: Example for creating temporal based features

$s_{id}$	$t$	$min(t)$	$\Delta time\_in\_days$	$a_i$	$a_j$	$s_{label}$
1	01.01.2013	01.01.2013	1	$a_{i_1}$	$a_{j_1}$	0
1	10.01.2013	01.01.2013	10	$a_{i_2}$	$a_{j_2}$	0
1	15.01.2013	01.01.2013	15	$a_{i_3}$	$a_{j_3}$	0
1	23.01.2013	01.01.2013	23	$a_{i_4}$	$a_{j_4}$	0
2	24.01.2013	24.01.2013	1	$a_{i_5}$	$a_{j_5}$	1
2	28.01.2013	24.01.2013	4	$a_{i_6}$	$a_{j_6}$	1
2	30.01.2013	24.01.2013	6	$a_{i_7}$	$a_{j_7}$	1

However, there are two directions of including the time for this feature construction technique. What we described above puts a stronger emphasis on the recent history. It is also possible to increase the weight of the past by using the  $(max\_date - current\_date)$  operator to calculate the  $\Delta time\_in\_days$  column.

The above mentioned techniques are applicable on numeric

```

Input:  $E$  // set of primitive numeric attributes
          $t$  // time indicator column
          $s_{label} \in \{0, 1\}$  // label indication
Def:  $a_i \in E$  // single attribute or a column in a data set
          $s_{id} = (r_1, r_2, \dots, r_m)$  // sequences of rows  $r$ 
          $S = \emptyset$  // set of constructed features
          $E_2 = E$  // copy of  $E$ , used for looping
          $ls$  // length of a sequence  $s_{id}$ 
          $max()$  // returns max value of a set
for each  $a_i \in E$  {
    for each  $s_{id}$  {
         $f_p = (\sum_{i=1}^{ls} ((\max_{k=1, \dots, ls} (t_k) - t_i) * a_i), s_{id}, s_{label})$ 
         $S = \{S \cup f_p\}$ 
    }
}
return  $S$ 

```

Fig. 4: Pseudo-code feature construction of temporal based attributes

attributes. For string attributes, it is possible to replace the string by the posterior probability  $p(\theta|x)$  (see also Hand [14], pp. 117-118 and pp. 354-356). Thereby  $\theta$  represents the probability of the parameters for a given evidence  $x$ . In our example case, we have the distribution of our two labels as parameters  $\theta$  and occurrences of  $a_i$  as evidence  $x$ . Based on this the posterior probability can be calculated as depicted in (5)

$$p(s_{label} = 1|a_i) = \frac{p(a_i|s_{label}=1)*p(s_{label}=1)}{p(a_i)} \quad (5)$$

In order to apply this for string based attributes, we can construct new features  $f$  for string attributes as depicted in (6)

$$f_p = \sum_{i=1}^{ls} (\max_{k=1, \dots, m} (t_k) - t_i) * (p(s_{label} = 1|a_i)) \quad (6)$$

If there are occurrences in the data that have a great tendency towards a particular label (i.e., having a high possibility for one label), we can make this pattern visible by multiplying the posterior possibility with the temporal axis of the given sequence.

However, if there are too many different occurrences, let's say more than 1.000 different values per attribute, this technique could have problems dealing with very small probabilities. So, it is recommended to take the logarithm of the posterior probability for cases with high cardinality.

## V. EXPERIMENTAL SETUP AND RESULTS

This section is divided into three subsection in which we will first look at the technical framework we used during our experiments. This is followed by a brief look at the data profile and the corresponding classification task. The third subsection will then compare and discuss the results of our experiments.

### A. Technical Framework and Infrastructure

All implementations and experiments were carried out on a Microsoft Windows Server 2008 R2 Enterprise Edition (6.1.7601 Service Pack 1 Build 7601) with four Intel Xeon CPUs E5320 (1.86 GHz, 1862 MHz). The available RAM comprised of 20 GB installed physical memory and 62 GB virtual memory (size of page file 42 GB). The widespread freeware data mining software RapidMiner (version 5.2.008) was used for the standard methods under comparison: Decision Tree, Naive Bayes, Neuronal Network and Random Forrest (for a closer description please also see Witten [13] pp. 191-294, Han [1] pp. 291-337). The method Bayesian Belief Network required the installation of the free RapidMiner extension WEKA. We used the default parameters for all of the above mentioned classification algorithms.

### B. Data Profile

The data we used for our experiments was retrieved from the DataMiningCup 2013. The training as well as the test data set can be downloaded on the following site: 'http://www.data-mining-cup.de/en/review/dmc-2013/'. The given historical data from an online shop consisting of session activities from customers. The goal of the task is to classify sessions into a buyer and a non-buyer class. The given training data has the following parameters:

- total number of rows: 429,013
- number of sessions: 50,000
- number of numeric attributes: 21
- number of string attributes: 2

The test data has the following parameters:

- total number of rows: 45,068
- number of sessions: 5,111
- number of numeric attributes: 21
- number of string attributes: 2

Most of the given attributes are numeric. Please note that there is no exact time column given. Therefore, we used a artificial  $id$  column to map the temporal order of the various sessions. We also used this column to calculate the temporal based features described in Subsection IV-D.

### C. Comparison of original attributes vs constructed features sets

As a first step, we used the given primitive attributes to solve the task. We used the accuracy measurement (7) due to a similar label distribution (45 % to 55 %) and both labels are associated with the same 'costs' for misclassification.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (7)$$

As it can be seen in Fig. 6, the Naive Bayes classification algorithm was able to achieve better result than the base line (the other algorithms defaulted and predicted label = 0 for all sessions). The Bayesian Belief Networks are not applicable for situations in which the same  $s_{id}$  can occur several time (therefore a accuracy rate of 0 %). In a next step, we used our

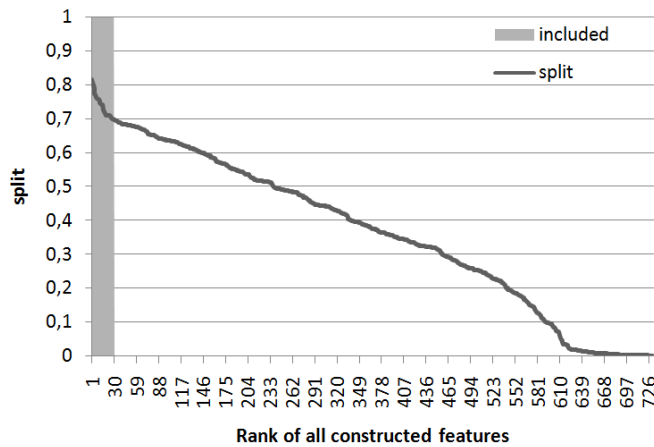


Fig. 5: All constructed features ranked by their split value.

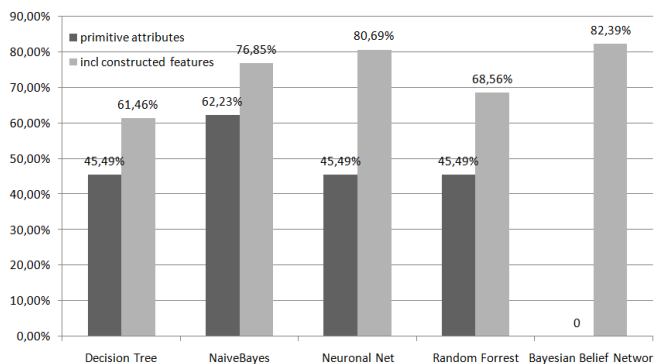


Fig. 6: Accuracy rate comparison original data set with primitive attributes and the same algorithms applied including the top 29 constructed features.

suggested feature construction algorithm in order to aggregate the sessions and find useful features. During this process, a grand total of 732 features were created:

- # of distinct occurrences based features: 23
- # of string concatenation based features: 2
- # of arithmetic based features: 686
- # of temporal axis based features: 21

All features were normalized with the min-max normalization and assessed by calculating the split value for each feature. The features were ranked by their split value, as it can be seen in Fig. 5. The best feature achieve a split value of 0.815, the lowest of 0.0003. In order to keep execution times low, we chose only the top 29 constructed features for our second run. Fig. 6 shows the impressive improvement for the compared standard methods. Since the  $s_{id}$  is unique for the constructed features set, the Bayesian Belief Networks are applicable.

### VI. CONCLUSION

Data pre-processing and selection are very important steps in the data mining process. This can be challenging, if there is no domain expert knowledge available. The algorithm proposed in this work helps, not only to understand the

patterns within the data, but also, to simplify more complex data structures (such as sequential data). It can be applied in conjunction with well known standard algorithms and can boost classification performance in a big variety of fields with similar specifications (such as the detection of credit card fraud, network intrusions, bots in computer games). Its systematic approach can also help domain experts to find previously unknown interactions among data and therefore, to get a better understanding of their domain.

### VII. FUTURE WORK

Further ways for extending the features space could be to implement more numerical features generated by logarithm, exponential powers or combinations of more than two attributes. The algorithm itself could be optimized to assess the quality of a candidate feature before actually calculating it. Another development direction could be to align the constructed features in a way, that would allow to classify data without the help of one of the standard algorithms.

### REFERENCES

- [1] J. Han and M. Kamber, "Data mining: Concepts and techniques" 2. edition pp. 48-97 second edition, San Francisco, Morgan Kaufmann, 2006
- [2] H. Liu and H. Motoda, "Feature Extraction, Construction and Selection: A Data Mining Perspective", Boston, Kluwe Academic Publisher, 1998
- [3] L. S. Shafiti and E. Pérez "Constructive Induction and Genetic Algorithms for Learning Concepts with Complex Interaction", in Proceedings of The Genetic and Evolutionary Computation Conference, Washington, June 2005, pp. 1811-1818
- [4] L. S. Shafiti and E. Pérez "Data Reduction by Genetic Algorithms and Non-algebraic Feature Construction: a Case Study", in Proceedings of: Eighth International Conference on Hybrid Intelligent Systems, Barcelona, September 2008, pp. 573-578
- [5] R. Setiono and H. Liu "Fragmentation Problem and Automated Feature Construction", in Proceedings of: fourth Conference on Data Mining and Optimization (DMO), Langkawi, September 2012, pp. 53-58
- [6] G. Pagallo "Learning DNF by Decision Trees", Machine Learning, pp. 71-99 Kluwer Academic Publishers, 1990
- [7] B. Zupan and M. Bohanec "Feature Transformation by Function Decomposition", in Journal IEEE Intelligent Systems archive Volume 13 Issue 2, March 1998, pp. 38-43
- [8] J.R. Quinlan "C4.5: Programs for Machine Learning". Morgan Kaufmann, 1993
- [9] D. García, A. González and R. Pérez, "A Two-Step Approach of Feature Construction for a Genetic Learning Algorithm", in Proceedings of: IEEE International Conference on Fuzzy Systems, Taipei, June 2011, pp. 1255-1262
- [10] D. García, Antonio González and R. Pérez, "An Iterative Strategy for Feature Construction on a Fuzzy Rule-Based Learning Algorithm", in Proceedings of: 11th International Conference on Intelligent Systems Design and Applications, Cordoba, November 2011, pp. 1235-1240
- [11] R. Alfred "DARA: Data Summarisation with Feature Construction", in Proceedings of: Second Asia International Conference on Modelling & Simulation, Kuala Lumpur, May 2008, pp. 830-835
- [12] F. Sia and R. Alfred "Evolutionary-Based Feature Construction with Substitution for Data Summarization using DARA", in Proceedings of: fourth Conference on Data Mining and Optimization (DMO), Langkawi, September 2012, pp. 53-58
- [13] I. Witten and F. Eibe, "Data mining : practical machine learning tools and techniques" 2. edition, San Francisco, Morgan Kaufmann, 2005, pp. 48-97
- [14] D. Hand, H. Mannila and P. Smyth "Principles of Data Mining", MIT Press, 2001

# Information Integration with Uncertainty: Performance

Inder K. Dumpa      Raja S. Kota      Fereidoon Sadri

Department of Computer Science

University of North Carolina at Greensboro

Greensboro, NC, USA

ikdumpa@uncg.edu, raja\_k999@yahoo.com, f\_sadri@uncg.edu

**Abstract**—Information integration and modeling and management of uncertain information have been active research areas for decades, with both areas receiving significant renewed interest in recent years. Research on information integration with *uncertainty*, on the other hand, is quite recent. In this paper we concentrate on recent works on uncertain-data integration. We present experimental results on the efficiency of recent algorithms for information integration from sources that contain uncertain data. Our experiments show the algorithms to be efficient, demonstrating a near linear performance in the total size of the uncertain data to be integrated.

**Keywords:** *Information integration; uncertain data; possible worlds; integration performance.*

## I. INTRODUCTION

The importance of information integration *with uncertainty*, has been realized recently [1]-[4]. It has been observed that [2]: “While in traditional database management managing *uncertainty and lineage seems like a nice feature, in data integration it becomes a necessity.*” Research on information integration with uncertainty, is quite recent [5]-[11]. Researchers have concentrated on two main aspects of information integration with uncertainty. The first category considers integration of definite data with uncertain schema mappings [6], [7]. The second category considers integration of uncertain data [5], [8], [9], [11]. Our work in [11] falls in the second category. We presented algorithms for the integration of uncertain data, and justified the correctness of the algorithms by showing they coincided with the integration formalism presented in the foundational work [5]. In this paper, we report the implementation and experimental results of these algorithms. Note that information integration has many dimensions and involves a number of important tasks such as data cleaning, data linkage, schema mapping, data standardization, query translation, and query optimization. We concentrate only on the issue of information integration from sources that contain uncertain data in this paper.

This paper is organized as follows. In Section II, we present an introduction to integration of information from sources with uncertain data, and briefly discuss the uncertain data integration algorithms of [11]. Our implementation is presented in Section III, and experimental results in Section IV. Our experiments show the algorithms to be efficient, demonstrating a near linear performance in the total size of the uncertain data to be integrated. Conclusion and future work are presented in Section V.

## II. PRELIMINARIES

In this section we will review some of the recent works regarding issues and algorithms for the integration of uncertain data.

### A. Information Integration with Uncertainty: Foundations

Foundations of information integration with uncertainty have been discussed in [5], [11]. We will present a brief summary here. We begin with an example from [11].

*Example 1:* John and Jane are talking about fellow student Bob. John says “I am taking CS100 and CS101, and Bob is in one of them, but not in both.” Jane says “I am taking CS101 and CS102 and Bob is in one of them, but not in both.”

Intuitively, if we integrate the information from these two sources (John and Jane), we should infer that Bob is either taking CS101, or he is taking both CS100 and CS102.

The model used in [5], [11] for the representation of uncertain information is the well-known *possible-worlds* model [12]. In Example 1, the information presented by the two sources (John and Jane) is represented by the possible worlds shown in Figures 1 and 2. The possible worlds of the result of integration is shown in Figure 3.

student	course	student	course
Bob	CS100	Bob	CS101

D1                      D2

Figure 1: Possible Worlds of source S1

student	course	student	course
Bob	CS101	Bob	CS102

D3                      D4

Figure 2: Possible Worlds of source S2

student	course	student	course
Bob	CS101	Bob	CS100
		Bob	CS102

Figure 3: Possible Worlds of the Integration for Example 1

Here, we will summarize the integration approach from [11] which uses a simple logic-based technique. It has been



shown to be equivalent to the integration formalism of [5] which is based on the concept of superset-containment. Interested readers are referred to [11] for details.

First, we should point out that the pure possible world model is not adequate for uncertain-data integration applications. We need additional information, namely, the set of all tuples. The following example demonstrates the possible-worlds with tuple sets model.

*Example 2:* Andy and Jane are talking about fellow student Bob. Andy says “I am taking CS100, CS101, and CS102 and Bob is in either CS100 or CS101 but not in both.” Jane says “I am taking CS101 and CS102 and Bob is in one of them, but not in both.”

Intuitively, if we integrate the information from these two sources, we should infer that Bob is taking CS101. The second possibility from Example 1 is not valid anymore since Andy’s statement rules out the possibility that Bob is taking 102.

To justify this answer, we observe that pairwise combination of possible worlds from the two sources result in the four possible worlds of Figure 4. But only the second possible world is a valid combination, and the other three are not valid. The first world is not valid since Andy states that he is taking CS100, CS101, and CS102 and Bob is taking 100 of 101 but not both. So Bob can not be in both 100 and 101. The third and fourth worlds are not valid due to Andy’s statement too. He is taking 102 (among other courses) and states that Bob is taking 100 or 101. Hence Bob can not be in 102. Note that the last world is also not valid due to Jane’s statements. She says that she is in 101 and 102, and Bob is in one of them, but not both. The only valid combination is the second world: Bob must be taking CS101.

student	course
Bob	CS100
Bob	CS101

student	course
Bob	CS101

student	course
Bob	CS100
Bob	CS102

student	course
Bob	CS101
Bob	CS102

Figure 4: Pairwise combination of possible worlds from the two sources

However, the possible-worlds representations of these sources (Andy and Jane) are exactly the same as those of Example 1 (Figures 1 and 2). Only when we add the tuple-set to possible worlds of Andy, namely  $\{(Bob, CS100), (Bob, CS101), (Bob, CS102)\}$ , It becomes explicit that Andy’s statement eliminates the possibility that Bob is taking CS102.

Hence, we will use the following definition from [5] for uncertain databases that adds tuple sets to the possible-worlds model. Note that to simplify presentation, it is assumed that possible worlds are sets of tuples in a single relation. We adopt the same convention throughout this paper.

*Definition 1:* (UNCERTAIN DATABASE). An *uncertain database*  $U$  consists of a finite set of tuples  $T(U)$  and a nonempty set of possible worlds  $PW(U) = \{D_1, \dots, D_m\}$ , where each  $D_i \subseteq T(U)$  is a certain database.

## B. Integration Using Logical Representation

The following definitions and results are from [11].

Given an uncertain database  $U$ , we assign a propositional variable  $x_i$  to each tuple  $t_i \in T(U)$ . We define the formula  $f_j$  corresponding to a possible world  $D_j$ , and the formula  $f$  corresponding to the uncertain database  $U$  as follows:

*Definition 2:* (LOGICAL REPRESENTATION OF AN UNCERTAIN DATABASE). Let  $D_j$  be a database in the possible worlds of uncertain Database  $U$ . Construct a formula as the conjunction of all variables  $x_i$  where the corresponding tuple  $t_i$  is in  $D_j$ , and the conjunction of  $\neg x_i$  where the corresponding tuple  $t_i$  is not in  $D_j$ . That is,

$$f_j = \bigwedge_{t_i \in D_j} x_i \bigwedge_{t_i \notin D_j} \neg x_i \quad (1)$$

The formula corresponding to the uncertain database  $U$  is the disjunction of the formulas corresponding to the possible worlds of  $U$ . That is,

$$f = \bigvee_{D_j \in PW(U)} f_j \quad (2)$$

Now we can integrate uncertain databases using their logical representations as follows:

Let  $S_1, \dots, S_n$  be sources containing (uncertain) databases  $U_1, \dots, U_n$ . Let the propositional formulas corresponding to  $U_1, \dots, U_n$  be  $f_1, \dots, f_n$ . We obtain the formula  $f$  corresponding to the uncertain database resulting from integrating  $U_1, \dots, U_n$  by conjuncting the formulas of the databases:  $f = f_1 \wedge \dots \wedge f_n$ .

*Example 3:* (INTEGRATION USING LOGICAL REPRESENTATION) Consider Example 1. The uncertain database corresponding to John’s statement is represented by  $(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$ , where  $x_1$ , and  $x_2$  correspond to the tuples (Bob, CS100) and (Bob, CS101), respectively. The uncertain database corresponding to Jane’s statement is represented by  $(x_2 \wedge \neg x_3) \vee (\neg x_2 \wedge x_3)$ , where  $x_2$  is as above and  $x_3$  corresponds to the tuple (Bob, CS102). The integration in this case is obtained as

$$\begin{aligned} & ((x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)) \wedge ((x_2 \wedge \neg x_3) \vee (\neg x_2 \wedge x_3)) \\ & = (x_1 \wedge \neg x_2 \wedge x_3) \vee (\neg x_1 \wedge x_2 \wedge \neg x_3) \end{aligned}$$

which corresponds to the possible worlds of Figure 3. The result is consistent with our intuition: Based on statements by John and Jan, Bob is taking either CS101 or both CS100 and CS102.

Now consider Example 2. The uncertain database corresponding to Andy’s statement is represented by  $(x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge x_2 \wedge \neg x_3)$ , where  $x_1$ ,  $x_2$ , and  $x_3$  represent (Bob, CS100), (Bob, CS101), and (Bob, CS102), respectively. The uncertain database corresponding to Jane’s statement is represented by  $(x_2 \wedge \neg x_3) \vee (\neg x_2 \wedge x_3)$ . The integration in this case is obtained as

$$\begin{aligned} & ((x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge x_2 \wedge \neg x_3)) \wedge \\ & \quad ((x_2 \wedge \neg x_3) \vee (\neg x_2 \wedge x_3)) \\ & = (\neg x_1 \wedge x_2 \wedge \neg x_3) \end{aligned}$$

corresponding to the (in this case, definite) relation consisting only of the tuple (Bob, CS101) (Figure 5). Again, this result is consistent with our intuition: Based on statements by Andy and Jane, Bob is taking CS101.

student	course
Bob	CS101

Figure 5: Possible Worlds of the Integration for Example 2

### C. An Alternative View of Integration

Let  $S_1, \dots, S_n$  be sources containing (uncertain) databases  $U_1, \dots, U_n$ . Let  $PW(U_i)$  represent the set of possible worlds of uncertain database  $U_i$ , and  $T_i$  represent the tuple set of  $U_i$ . We can regard the integration of information from these sources as follows:

*Definition 3: (COMPATIBLE SET OF POSSIBLE-WORLDS RELATIONS).* Consider a set of  $n$  relations  $\{w_1, \dots, w_n\}$  where each  $w_i$  is a relation in the set of possible worlds of  $U_i$ , that is,  $w_i \in PW(U_i)$ ,  $i = 1, \dots, n$ . If there is a tuple  $t$  in a relation  $w_i$ , that it is also in  $T_j - w_j$  for some other possible-world relation  $w_j$ , we say the set of possible-world relations  $\{w_1, \dots, w_n\}$  is *not compatible*. Otherwise,  $\{w_1, \dots, w_n\}$  is *compatible*.

Note that  $t \in T_j - w_j$  means that according to source  $S_j$ , the tuple  $t$  can not exist (is ruled out) in  $w_j$ . Hence, if a set of possible world-relations is not compatible, they can not be integrated. A compatible set of possible-world relations  $\{w_1, \dots, w_n\}$  can be integrated, and the resulting relation contains all the tuples in the relations, that is, the result of integrating  $w_1, \dots, w_n$  is  $w = \cup_{i=1}^n w_i$ .

Hence, to integrate sources  $S_1, \dots, S_n$ , we can compute the possible-worlds relations of the integration by

- 1) forming all possible combinations  $\{w_1, \dots, w_n\}$ ,  $w_i \in PW(U_i)$ ,
- 2) determining compatible sets, and
- 3) obtaining the union of the relations in the compatible set.

This alternative characterization of integration results in a simpler integration algorithm. We use the logical formulation only to determine compatible sets of possible worlds, and then we obtain the result by calculating the union of the possible worlds in each compatible set. We have used this characterization to design our integration algorithm (Section III).

*Example 4: (ALTERNATIVE VIEW OF INTEGRATION)* Consider Example 1. The possible-worlds relations of the uncertain database corresponding to John's statement were shown in Figure 1, and the possible-worlds relations of the uncertain database corresponding to Jane's statement were shown in Figure 2. In this case, the compatible sets of possible worlds are  $\{D_1, D_4\}$  and  $\{D_2, D_3\}$ . We can conveniently represent the compatibility of possible-worlds relations for two sources by a bi-partite graph, such as Figure 6. The possible-worlds of the result of integration is shown in Figure 3.

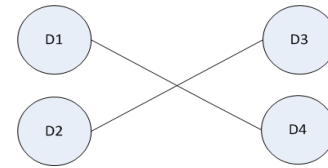


Figure 6: Compatibility graph of Example 1.

## III. IMPLEMENTATION

We implemented the information integration approach of Section II-C, and ran a large number of experiments to assess the performance of the implementation. In this section we present the implementation details. Experimental results are presented and discussed in the next section.

Our implementation consists of several modules implemented in Java.

- The `GeneratePossibleWorlds` module is a utility module used to randomly generate possible world relations for the information sources. The user can specify the following parameters:
  - Number of information sources.
  - Number of possible worlds for each source.
  - Number of tuples for each possible world.
  - Number of attributes for the possible-worlds relations.

To generate random tuples for possible worlds relations, we stored several files of domain values. Each file contains a large number of values from a specific domain, such as names, course numbers, course titles, semesters, and years. The user can specify the number of attributes for the possible world relations. The system forms a random tuple by randomly picking values from the domain of each attribute. The `GeneratePossibleWorlds` module stores the possible worlds relations in Oracle databases, one for each source. The sizes of the relations, and the total size of the integration instance are also recorded.

These features were used to generate desired cases for the experiments by altering the number of sources, number of possible worlds, size of each possible world, and total size of the sources data. Hence we can evaluate the impact of each parameter on the performance of the algorithm.

- The `TableIntegration` module performs the following tasks for each dataset generated by the `GeneratePossibleWorlds` module discussed above.
  - The module accesses the possible world relations in the databases of the sources. Each source is represented by an Oracle database that contains the possible world relations for that source.
  - The tuple set for each source is computed as the union of the possible world relations for that source.
  - The module generates the logical formula for each possible world relation for the sources

according to the algorithm of Section II-B. The formulas are conveniently represented by vectors, which can be used to easily implement logical operations over the expressions.

- The module determines which sets of possible world relations, one from each source, are *compatible* and hence can produce a possible world relation in the integration. This is done by computing the conjunction (logical and) of the corresponding formulas of the possible world relations. If the result is *false*, the set of possible worlds are not compatible. Otherwise they are compatible.
- For all compatible sets of possible worlds, the modul generates the resulting relation by unioning the possible worlds relations in the set. It stores the integrated relation in the Oracle database for the integration result.
- Once all compatible possible wrolds sets are processed, the module displays the total time for the integration.

#### IV. EXPERIMENTAL EVALUATION

We carried out a large number of experiments to evaluate the performance of the integration algorithm. The experiments were executed on a 2.10 GHz Intel i3 CPU with 4.00 GB RAM, 64-bit Windows 7 Operating System using Java 1.7 and Oracle XE 10g. The first few experiments evaluated the performance of the integration algorithm for integrating information from two sources.

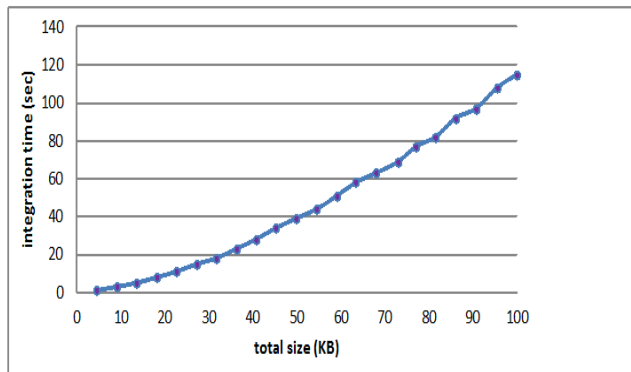


Figure 7: integration times; fixed number of possible worlds for each source.

In the first set of experiments the number of possible world relations of the two sources were kept constant, and test cases were generated by varying the number of tuples in the possible world relations (and hence, varying the size of uncertain databases to be integrated). Figure 7 shows the result of these experiments. The horizontal axis shows the total size (KB) of databases to be integrated. The vertical axis shows the time needed for the integration (sec). The experiments show that the integration algorithm is almost linear in the total size of databases to be integrated.

In the second set of experiments, we varied the number of possible world relations of the two sources while keeping the number of tuples constant. Figure 8 shows the result of these

experiments. The horizontal axis shows the total size (KB) of databases to be integrated. The vertical axis shows the time needed for the integration (sec). Again, the experiments show that the integration algorithm is almost linear in the total size of databases to be integrated (no matter whether the size increase is due to larger number of possible worlds per sources, or larger possible world relation sizes.)

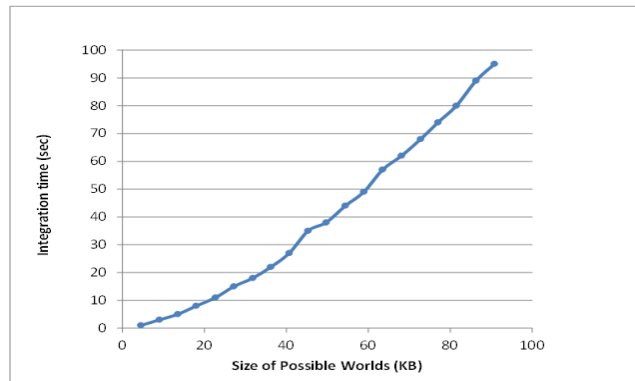


Figure 8: integration times; variable number of possible worlds for each source.

In the next set of experiments we evaluated the impact of the number of possible world relations and their sizes on the integration algorithm. The total size was kept constant (approximately) by changing both the number of possible world relations and the number of tuples in these relations accordingly. The values of these parameters and the integration time are shown in Table I. The columns are, respectively, number of possible worlds for each source, number of tuples in each possible world, total size, and integration time. Total size is almost constant – it ranges between 99.6 and 100.4 KB.

TABLE I: Integration experiments; with total size (almost) constant

PWs	tuples	size	time
20	110	99.9	114
19	116	99.8	114
18	122	99.6	109
17	130	100.4	113
16	138	100.2	116
15	147	100.0	114
14	158	100.3	111
13	170	100.3	116
12	184	100.2	113
11	200	99.8	113
10	220	100.0	114
9	245	100.0	114
8	275	100.0	114
7	315	99.9	111
6	368	100.2	115
5	442	100.3	116
4	553	100.1	119
3	737	100.4	120

Figures 9 and 10 plot the integration time in the experiments of Table I against the number of possible worlds

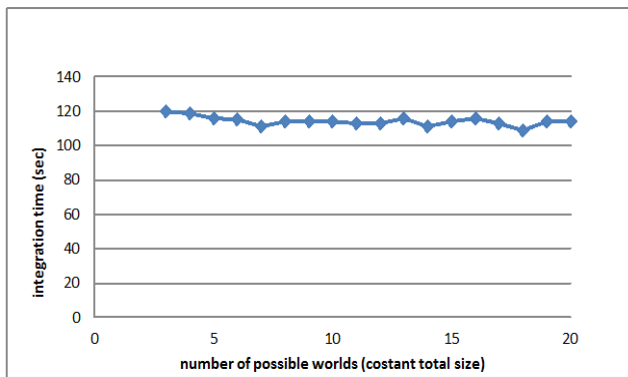


Figure 9: integration times vs number of possible worlds; total size is constant

and the number of tuples in each possible world. The x-axis of Figure 10 (number of tuples in each possible world) is logarithmic to better demonstrate the effect of number of tuples, ranging from 110 to 737 in the experiments. These experiments show that the number of possible worlds and their sizes are not factors in the performance of the integration algorithm when the total size is constant. In other words, integration time is almost constant when number of possible world relations and their sizes change while the total size is fixed. This observation is counter-intuitive since the integration algorithm needs to determine, for every pair of possible worlds  $(w_1, w_2)$ , whether they are compatible, where  $w_1$  and  $w_2$  belong to source 1 and source 2, respectively. But the impact of number of tuples (smaller number of tuples for larger number of possible world relations) counterbalances the impact of number of possible worlds.

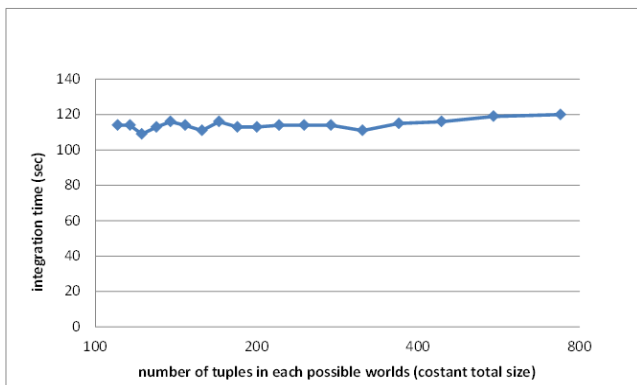


Figure 10: integration times vs number of tuples in each possible world; total size is constant

In the next set of experiments we generated test cases by varying both the number of possible worlds and the number of tuples in each possible worlds (and hence, varying the total size). The results are summarized in Table II. The columns are, respectively, number of possible worlds for each source, number of tuples in each possible world, total size, and integration time. Figure 11 plots the integration time in these experiments against total size. It confirms a near linear performance of the algorithms as a function of the total size of the integration. Figure 12 plots the integration time against the

number of tuples in each possible-world relation. Note that the same figure is also the plot of integration time against number of possible worlds, since we are using the same numbers for the two parameters for each data point (See Table II). This figure suggests integration time is a quadratic function of number of possible worlds (or number of tuples in each possible world). This is no surprise, since by varying both these parameters (and using the same numbers) we obtain a total size that is quadratic in each of these parameters. So, again, we confirm that total size is the important parameter in the performance of the algorithm.

TABLE II: Integration experiments; varying number of possible worlds and number of tuples in each possible world

PWs	tuples	size	time
4	4	0.7	0
8	8	2.9	1
12	12	6.5	3
16	16	11.5	4
20	20	18	7
24	24	26.1	13
28	28	35.4	22
32	32	46.4	33
36	36	58.8	50
40	40	72.7	70
44	44	87.8	89
48	48	104.5	121
52	52	122.5	149
56	56	142	182
60	60	163.4	219

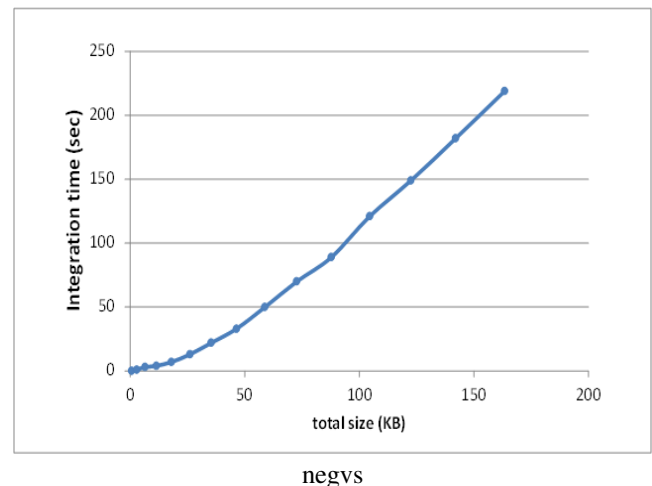


Figure 11: integration times vs total time

In the next set of experiments we evaluate the performance of the integration algorithm when integrating data from more than two information sources. We generated test cases by varying the number of information sources, while keeping the total size constant. Figure 13 plots the integration time against the number of possible worlds. This performance was very unexpected. As seen from this graph, the algorithm has an almost constant time up to about 10 information sources, then the integration time increases sharply. We postulated that the reason for the sharp increase is memory saturation, which

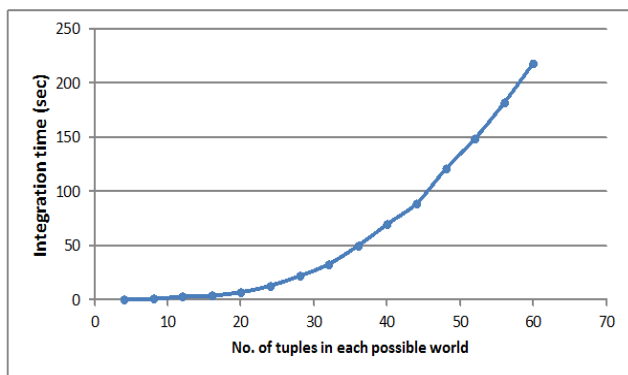


Figure 12: integration times vs number of tuples in possible-world relations

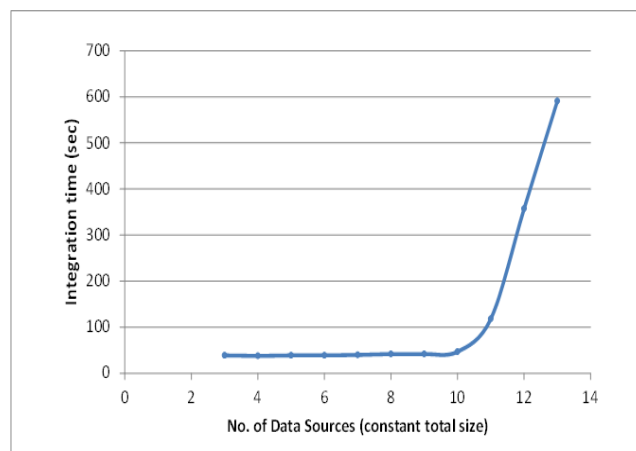


Figure 13: integration times vs number of sources

forces the execution to use virtual memory. In our experiments, the number of possible world relations for each source was kept constant at 3, and constant total size was achieved by varying the number of tuples in each possible world. So, with 10 sources, the number of integration combinations to generate was  $3^{10} = 59,049$ . This number increases to  $3^{11} = 177,147$  for 11 sources, and to  $3^{12} = 531,441$  for 12 sources. The memory of our testbed system saturates at about 10 sources.

To test our hypothesis, we executed the exact same experiments on systems with lower (2GB) and higher (8GB) memory sizes. The graphs for these experiments have the same shape, except at lower memory size the graph is shifted to the left, and at higher memory size the graph is shifted to the right. In other words, the sharp increase happens at a lower number of sources for the lower memory size, and at a higher number of sources for higher memory size. These additional experiments confirm our hypothesis that the change in the performance of the integration algorithm, from constant time to almost linear, is a result of memory saturation. So, our final conclusion is that, given adequate memory, the performance of the integration algorithm is a linear function (approximately) of the total size of the integration instance. It is not sensitive to the other factors, namely, number of information sources, number of possible worlds relations in the sources, and number of tuples in the possible world relation, when the total size is kept constant.

## V. CONCLUSION

We presented our implementation and experimental evaluation of the uncertain-data integration algorithms of [11]. Our experiments show the algorithms to be efficient, demonstrating a near linear performance in the total size of the uncertain data to be integrated.

There are a number of important issues that require further investigation. First, uncertain schema mappings is another source of uncertainty in information integration. We would like to develop integration algorithms for this case, with definite or uncertain data. The integration algorithm is a good candidate for parallel computation, in particular, using the map-reduce framework [13]. A future direction would be to implement the integration using Hadoop running on a large number of computers. More importantly, we would like to devise

integration algorithms to work with compact representations of uncertain data, such as the probabilistic relational model of [14], [15].

## REFERENCES

- [1] L. M. Haas, "Beauty and the beast: The theory and practice of information integration," in Proceedings of International Conference on Database Theory, 2007, pp. 28–43.
- [2] A. Y. Halevy, A. Rajaraman, and J. Ordille, "Data integration: The teenage years," Proceedings of International Conference on Very Large Databases, 2006, pp. 9–16.
- [3] M. Magnani and D. Montesi, "Uncertainty in data integration: current approaches and open problems," in Proceedings of VLDB Workshop on Management of Uncertain Data, 2007, pp. 18–32.
- [4] —, "A survey on uncertainty management in data integration," ACM Journal of Data and Information Quality, vol. 2, no. 1, 2010.
- [5] P. Agrawal, A. D. Sarma, J. D. Ullman, and J. Widom, "Foundations of uncertain-data integration," Proceedings of the VLDB Endowment, vol. 3, no. 1, 2010, pp. 1080–1090.
- [6] X. L. Dong, A. Halevy, and C. Yu, "Data integration with uncertainty," in Proceedings of International Conference on Very Large Databases, 2007, pp. 687–698.
- [7] X. L. Dong, A. Y. Halevy, and C. Yu, "Data integration with uncertainty," The VLDB Journal, vol. 18, no. 2, 2009, pp. 469–500.
- [8] A. A. Eshawi and F. Sadri, "Information integration with uncertainty," in Proceedings of International Database Engineering and Applications, IDEAS, 2009, pp. 284–291.
- [9] R. Fagin, B. Kimelfeld, and P. G. Kolaitis, "Probabilistic data exchange," Journal of the ACM, vol. 58, no. 4, 2011, p. 15.
- [10] D. Florescu, D. Koller, and A. Y. Levy, "Using probabilistic information in data integration," in Proceedings of International Conference on Very Large Databases, 1997, pp. 216–225.
- [11] F. Sadri, "On the foundations of probabilistic information integration," in Proceedings of International Conference on Information and Knowledge Management, 2012, pp. 882–891.
- [12] S. Abiteboul, P. C. Kanellakis, and G. Grahne, "On the representation and querying of sets of possible worlds," in Proceedings of ACM SIGMOD International Conference on Management of Data, 1987, pp. 34–48.
- [13] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in Proceedings of Operating System Design and Implementation, 2004, pp. 137–150.
- [14] N. N. Dalvi and D. Suciu, "Efficient query evaluation on probabilistic databases," in Proceedings of International Conference on Very Large Databases, 2004, pp. 864–875.
- [15] —, "Efficient query evaluation on probabilistic databases," The VLDB Journal, vol. 16, no. 4, 2007, pp. 523–544.

## Universal Evaluation System Data Quality

María del Pilar Angeles, Francisco Javier García-Ugalde,  
 Carlos Ortiz, Ricardo Valencia, Eduardo Reyes,  
 Arturo Nava, Jhovany Pelcastre  
 Facultad de Ingeniería  
 Universidad Nacional Autónoma de México  
 México, D.F.

pilarang@unam.mx, fgarciau@unam.mx,  
 carlos.ortiz.eng@comunidad.unam.mx, ricardofdk8@hotmail.com, eduardorv07@hotmail.com,  
 arturo\_shox@hotmail.com, j.pelcastre@hotmail.com

**Abstract**— This paper presents a work in progress regarding to the extension and improvement of the Freely Extensible Biomedical Record Linkage system. Currently, the prototype has been extended to directly connect to a number of database managements systems, calculate their database quality indicators, automatically generate a flat file from any database and execute an appropriate data matching process.

*Keywords*-data matching; de-duplication; record linkage

### I. INTRODUCTION

When an enterprise information system is meant to be built upon integration of their existing heterogeneous database systems, they would face the difficulty of comparing disparate schemas in order to identify syntactic and semantic heterogeneities; make these schemas correspond and match them through transformation functions; and finally, comparing data of unknown quality such as name, address from a single record against a large number of records.

Integrating data from different sources consists of three tasks [1]. The first task is concerned with identifying database tables, attributes and conceptual structures from disparate databases that contain data that correspond to the same type of information, namely schema matching [2]. The second task is concerned with the identification and match of records that correspond to the same entity, when they come from disparate data sources, called data matching. In the case of identification of records that actually refer to the same entity within a single database, is known as duplicate detection [1]. Duplicated records can be handled in different ways, providing the complete set of inconsistent answers, providing the complete set of answers, but ranked according to likelihood of being correct [3], providing a single value selected at random, providing a top value in a ranked answer, or providing a fused answer [4], which is the process of merging pairs or groups of records that have been classified as matches into a clean and consistent record that represents an entity. When applied on one database, this process is called de-duplication. We assume that the process of schema matching has already achieved.

The open issues on data matching are mainly concerned to the record comparison among databases in order to determine if a pair of records corresponds to the same entity or not, because the process grows exponentially as the

databases to be matched get larger. In real-world data matching applications, the true status of two records that are matched across two databases is not known. Therefore, accurately assessing data matching quality and completeness is challenging [1].

This approach is aimed to the development of algorithms that reduce the quadratic complexity of the naive process of pair-wise comparing each record from one database with all records in the other database, and how to accurately classify the compared record pairs into matches and non-matches considering attributes dependency.

Nowadays, we are focused on the implementation of algorithms in order to measure, assess and help during the analysis of data quality process under a number of open and licensed database management system (DBMS), such as Oracle DB, MySQL, IBM DB2, SAP-Sybase Adaptive Server Enterprise, SAP-Sybase IQ, EnterpriseDB PostgreSQL.

For the process of identification, analysis and merge of duplicated records, we are still working on the extension of the Freely Available Record Linkage System (FEBRL) [5] developed by a research group of the Department of Computer Science at The Australian National University.

We are focused on the integration of the FEBRL system to any database from any DBMS by querying the native data dictionary; the research proposal is also aimed to the enhancement and addition of further standardization, indexing, and classification algorithms for data matching.

We have called our prototype as FEBRL-SEUCAD, it will support six DBMS at least. The application extracts the database schema directly from the data dictionary and measures the intrinsic quality of the data through the following indicators: coverage, density, completeness [6]. Since these measures are intrinsically computed through SQL queries, the assessed granularity levels are at database, table and column where applicable as we have done in previous research [7]. Furthermore, the prototype will implement a specific framework for the detection, classification and fusion (cleaning) of duplicate records within a number of databases (data matching and de-duplication) with no regard of the type of data source.

The present paper is organized as follows: The next section is focused on the assessment of data quality. The third section briefly explains the data matching process. Section IV describes the work we have carried out regarding

to the extension and enhancement of algorithms of the data matching process. The last section concludes the main topics achieved and the future work to be done.

## II. ASSESSMENT OF DATA QUALITY

The strictness of quality assessment is a weak or strong characterization depending on evaluating the quality property as a percentage or as a Boolean function respectively as shown in [8]. The strong characterization of the quality metrics is useful in applications where it is not possible to admit errors at the corresponding level of granularity.

In the case of the assessment of data quality, we have considered the weak strictness to make possible the comparison of data sources for a number of data quality properties in. However, there might be alternatives where strictness could depend on the level of quality required, according to specific applications.

In order to assess data quality at different levels of granularity [7], we have utilized the measures provided at lower levels of granularity (data value, attribute) to determine aggregated scores (table, database) as we move through the levels of granularity.

Regarding completeness, we have taken the corresponding metrics of [6] and [8] for the value, attribute, and relation granularity levels, and we have incorporated completeness at the database level.

**Coverage:** This is the measure for the number of tuples a source stores; in other words as the probability that an entity of the world is represented in the source [6]. This is also contemplated under the Open World Assumption without nulls completeness case, at the relation level of granularity, refer to [8] for further detail.

**Density of an attribute:** is the measure of how well the attributes stored at a source are filled with actual (non-null) values (columns), in [6], a weak attribute completeness case under the Closed World Assumption with nulls in [8].

Density of the source  $d(S)$  is obtained by the average density over all density attributes [6].

Weak relation completeness is the number of tuples with all its attributes filled with non-null values divided by the number of tuples [6].

The completeness at database level will correspond to the average completeness of its corresponding relations.

The measurements are given by the aggregation of values at each of these levels as they are moving on. As a measurement of data quality is directly related to the level of granularity, we conclude that scores measured at lower level of granularity will provide a greater degree of accuracy than aggregated scores produced at higher levels.

The functions utilized for aggregation of scores are commonly average, maximum, and minimum. The appropriateness of an aggregation function will depend on the optimistic, conservative, or pessimistic approach taken according with the application context. It is not our intension to identify the best aggregation function, because there is not an absolute value. As long as the aggregation function reflects the user needs and it is consistently used, it should be enough for the estimation of quality and comparison purposes.

## III. THE DATA MATCHING PROCESS

### A. Introduction

The data matching process in general terms is focused on joining records from one data source with another that describe the same entity. This process requires the following tasks: data standardization [9]; indexing possible matching data in order to reduce the number of comparisons; data comparison and classification of pairs of records in possible match, not match and match. These steps are briefly explained during this section.

### B. Standardization

The standardization process [9] refers to the conversion of input data from multiple databases into a format that allows correct and efficient record correspondence between two data sources. Within the first step, called tokenization, it is assumed that the attributes of the input databases contain values that are separated by spaces, known as tokens. The second step is concerned with the detection and correction of data values that contain typographical errors or variations already known. The third step is the segmentation of tokens in well-defined output fields for proper data mapping or identification and correction of duplicate values (known as de-duplication).

### C. Indexing

A detailed process of records comparison is usually computationally expensive. That is, the complexity is quadratic according to the length of the attribute values (mostly chains) that are correlated. The comparison process is the most complex of all the data mapping steps. The indexing aims to reduce the number of pairs of records that will be compared, reducing those pairs of records that are unlikely to correspond to the same real world entity and retaining those records that probably would correspond in the same block for comparison reducing the number of record comparisons. Therefore, the definition of the locking key is very important, because it will specify how to keep similar records in the same block of comparison. The record similarity depends on the data types they contain because they can be similar phonetically, numerically or textually. Some of the methods implemented within Febrl are for instance, Soundex [10], Phonex [1], Phonix [1], NYSIIS[11], Double metaphone [12], QGrams.

### D. Field and record Comparison Methods

As the comparison data might be of low quality (they may contain typographical errors or variations), establishing a binary or strict criterion for the comparison process such as (similar / dissimilar) is not possible or realistic. Therefore, the comparison methods implemented provide degrees of similarity and define thresholds depending on the semantics and data type of each field. Some of the methods implemented within Febrl are for instance, Qgram, Jaro - Winkler Distance [13], [14] Longest common substring Comparison.

### E. Classification

The classification of pairs of records grouped and compared in the previous steps [15], [6], is mainly based on the similarity values were obtained, since it is assumed that the more similar two records are, there is more probability that these records belong to the same entity of the real world.

The records are classified into “matches”, “not matches” or “possible matches”, the classification of records can be an unsupervised or supervised process.

The unsupervised process classifies pairs or groups of records in the similarities between them without having access to more information about the characteristics of those records.

The supervised process requires training based on data identified as similar or not similar. In this case, comparison vectors with an associated value that determines whether records correspond or not are required.

In the case of potentially corresponding records, the duplicates detection may be performed manually.

Within Febri, there are methods based on thresholds, probabilistic methods, costs based methods or rule-based methods.

The accuracy of data matching is mostly influenced by the comparison and classification steps. However, the indexing step will impact on the completeness of a data matching exercise because record pairs filtered out in the indexing step will be classified as non-matches without being compared.

The most commonly way to classify candidate record pairs is to sum the similarity values in their comparison vectors into a single total similarity value and to then apply two similarity thresholds to decide the class a candidate record pair belongs to. However, there are some dependencies between attributes. For instance, records with the same postcode will potentially have the same street name [1]. Therefore, if we assume that all similarity values are normalized between 0 and 1, all attribute similarities contribute in the same way towards the final summed similarity value. The importance of different attributes, as well as their discriminative power regard to distinguishing matches from non-matches, is not considered. Furthermore, with no regard of a weighted or an unweight approach, the detailed information contained in the individual similarity values is lost by such a simple summation approach.

For instance, the probabilistic classification approach of Fellegi and Sunter [6] is one of the most utilized nowadays because it allows the calculation of weights for corresponding and not corresponding pairs of attribute values, which leads to a better decision during records pair classification, but by assuming a conditional independence.

The present research is currently on the implementation of an enhancement to this issue.

### F. Evaluation of Matching

Matching quality refers to how many of the classified matches correspond to true real-world entities, while matching completeness is concerned with how many of the real-world entities that appear in both databases were correctly matched [17].

Each of the record pair corresponds to one of the following categories [18]:

- True positives (TP). These are the record pairs that have been classified as matches and that are true matches. These are the pairs where both records refer to the same entity.

- False positives (FP). These are the record pairs that have been classified as matches, but they are not true matches. The two records in these pairs refer to two different entities. The classifier has made a wrong decision with these record pairs. These pairs are also known as false matches.

- True negative (TN). These are the record pairs that have been classified as non-matches, and they are true non-matches. The two records in pairs in this category do refer to two different real-world entities.

- False negatives (FN). These are the record pairs that have been classified as non-matches, but they are actually true matches. The two records in these pairs refer to the same entity. The classifier has made a wrong decision with these record pairs. These pairs are also known as false non-matches.

An ideal outcome of a data matching project is to correctly classify as many of the true matches as true positives, while keeping both the number of false positives and false negatives small.

Precision calculates the proportion of how many of the classified matches (TP + FP) have been correctly classified as true matches (TP). It thus measures how precise a classifier is in classifying true matches. [19]. It is calculated as:  $\text{precision} = \text{TP} / (\text{TP} + \text{FP})$

Recall measures how many of the actual true matching record pairs have been correctly classified as matches [19]. It is calculated as:  $\text{recall} = \text{TP} / (\text{TP} + \text{FN})$ .

At the present time, we have been focused on the enhancement of the Fellegi and Sunter probabilistic classification in terms of keeping the match weight after classification.

### G. Related work

The FEBRL project has developed prototype software which undertakes data standardisation, which is an essential pre-processing phase for most record linkage projects, and which implements the "classical" approach to probabilistic record linkage model as described by Fellegi and Sunter in [16]. We are focused on the extension of the original FEBRL system to any database from any DBMS by querying the native data dictionary; the research proposal is also aimed to the enhancement and addition of further standardization, indexing, and classification algorithms for data matching.

We are currently analysing which de-duplication algorithms are suitable for incorporating to the FEBRL-SEUCAD in order to implement them and compare them to the already implemented on FEBRL.



#### IV. FEBRL-SEUCAD

This section presents the enhancement we have implemented to the original FEBRL project so far. However, as we have pointed out before, there is a long way of further work to be done.

The prototype FEBRL-SEUCAD is now able to connect to any application implemented under a number of Database Management Systems such as PostgreSQL, SAP Sybase Adaptive Server Enterprise, SAP Sybase IQ, MySQL, Oracle and IBM DB2 with only the name of the database to be analyzed as a parameter. FEBRL-SEUCAD contains code to extract the native data dictionary in order to obtain all the database objects created under such database name.

##### A. Activities

To develop the prototype we have undertaken the following activities.

- Extraction of database objects by the data dictionary from each DBMS.
- Implementation by SQL programming of quality metrics such as coverage, density and uniqueness, the latter considering primary key, because otherwise would be data de-duplication or data matching covered by Febrl. Such sql programming has been carried out for each DBMS in their corresponding SQL language.
- Extension of the Febrl application for connection to any database through the already mentioned DBMS.
- Extension of the Febrl application to incorporate the options concerned to compute the quality metrics at database, table, record and column.
- Extension of the Febrl application to incorporate the option of selecting a specific database object and its corresponding data matching process.

The prototype currently supports the measurement of data qualitative dimensions within a number of database management systems and the steps within the data matching process (indexing, comparison, and classification).

The application extracts the database schemas directly from the data dictionary and identifies the following indicators: coverage, density, complete and uniqueness, since they are intrinsically calculable through SQL, granularity levels are calculated at database, table, and column log for a number of Database Management Systems.

##### B. Operations

This section is aimed to briefly describe the operation of the FEBRL-SEUCAD prototype. In the case of Data profiling, the first step is to select the metrics option, specify the required metric, the level of granularity to compute, the Database management system and the database name. We have extended the Febrl system in order to calculate completeness and uniqueness at different levels of granularity by extracting from the data dictionary the database objects, this feature allows the prototype to be utilized on any database platform. Fig. 1 shows completeness at database level from a SAP-Sybase Adaptive Server Enterprise database.

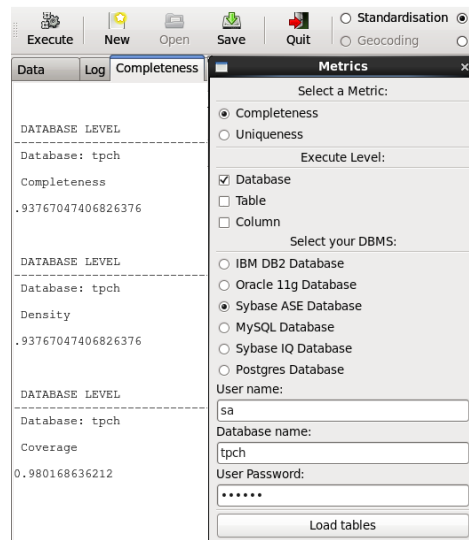


Figure 1 Completeness at database level

Once the database object has been selected, we have extended Febrl as shown in Fig. 2, in order to automatically generate the corresponding flat file in CSV, TBL formats, where originally the flat file had to be generated apart and then loaded to the application for further analysis.

rec_id	given_name	surname	street_number	address_1	address_2	suburb	pos
rec-3-dup-0	jeremy	frantiesk	12	nelumbo street		corrimal	210
rec-4-org	hannah	collier	28	banfield street	cara hill	sunshine beach	546
rec-0-dup-0	maddison	white	139	place colton		hocking	209
rec-4-dup-0	hannah	collier	2	banfield street	cara hill	sunshine beach	546
rec-0-org	maddison	white	139	colton place		hocking	209
rec-1-dup-0	james	mutavcic	115	gaunson crescent		woolgoonga	313
rec-2-org	laura	vlassopoulos	11	wilkins street	the gums	surfers paradise	504
rec-1-org	james	mutavcic	115	gaunson crescent		woolgoonga	313
rec-2-dup-0	laura	vlassopoulos	11	wilkins street	the gums	surfers paradise	504
rec-3-org	jeremy	frantiesk	12	nelumbo street		corrimal	210

Figure 2 Flat file generated from a database table

The data profiling step helps to determine the number of different data values for such attribute, the distance frequency, and the number of records with empty values. These brief data profiling allows the identification of a suitable attribute for indexing.

Indexing: The next step is the identification of attributes that would help on the execution of the indexing process along with specification of the corresponding parameters according to such indexing method. The best suited attributes for indexing are those with no missing values and uniform frequency distance. For instance, in the case of QGramIndex it is possible to specify the number of Q-grams and threshold as is shown in Fig. 3.

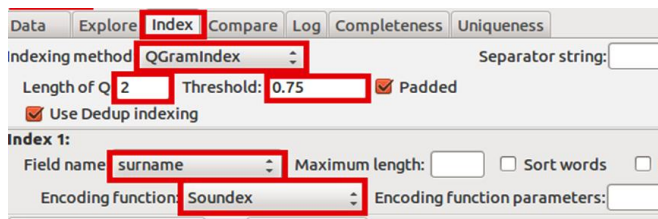


Figure 3 Indexing by QGramIndex and Soundex encoding

Comparison: The most common attributes for comparison are strings of one token, such as surname, family name, or a short number of tokens such as address, street name. Within SEUCAD-FEBRL is required to also specify the field comparison function per each comparison attribute. For instance, in Fig. 4, the comparison function Bag-Distance is used for surname and the Number-Percentage algorithm for the attribute street\_number.

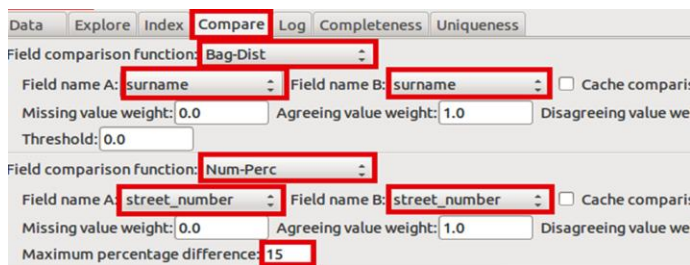


Figure 4 Comparison by Bag-Distance and Number percentage

Classification: The data type of the attributes involved for the classification process is relevant in order to specify the classification method. In Fig. 5 the classification method chosen is Fellegi Sunter with a lower threshold of 8 and an upper threshold of 1.8.



Figure 5 Classification by Fellegi and Sunter

Once identified the attributes and methods for each step, it is possible to execute the data matching process, which is the case shown in Fig. 6.

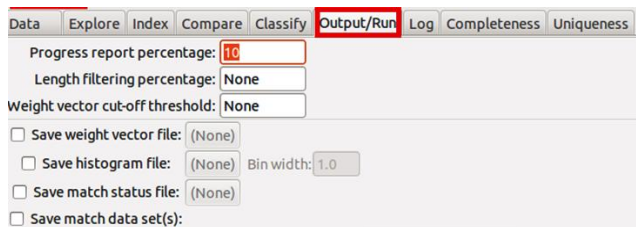


Figure 6 Execution of data matching process

Fig. 7 presents the number of matches, non-matches and possible matches as the outcome of the data matching process.

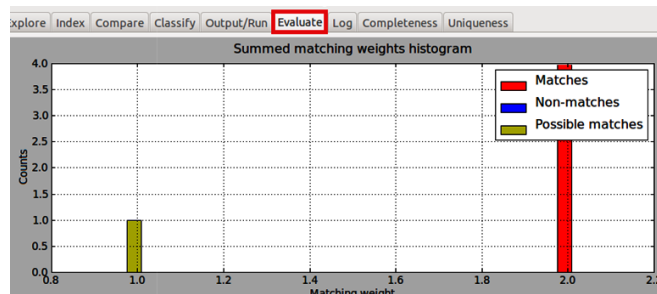


Figure 7 Outcome of matches, non matches and possible matches

The data matching outcome shows 4 matches, 1 possible match, there were not pair of records with non matches.

The evaluation of the data matching algorithms has already been coordinated by Peter Christen during the development of the FEBRL prototype [1], [5].

The data quality indicators: coverage, density, completeness and uniqueness have been identified in previous work [3], [6], [7], [8] and extended to different levels of granularity in [7]. However, as these quality indicators are an addition to the original de-duplication prototype, they have been calculated and tested at database, table, and column log for a number of Database Management Systems within FEBRL-SEUCAD.

## V. CONCLUSION AND FUTURE WORK

### A. Conclusion

We present a work in progress regarding the extension and improvement of an open software for de-duplication of records originally called FEBRL.

Currently, the FEBRL-SEUCAD prototype can directly connect to several database management systems such as Oracle DB, MySQL, IBM DB2, SAP-Sybase Adaptive Server Enterprise, SAP-Sybase IQ, EnterpriseDB PostgreSQL, extracted from the dictionary database objects of interest in order to allow data platform independency.

Our prototype is able to calculate data quality indicators allowing a better decision regarding to the identification of attributes that would help on the data matching process.

FEBRL-SEUCAD allows the flat file generation from any object database in CSV, TBL formats and then loaded to the application for further analysis.

### B. Future work

We are planning the enhancement of some of data matching algorithms already implemented in the original FEBRL prototype, at the present time; we are focused on the enhancement of the classification process by considering the importance of different attributes through their corresponding weights.

The implementation of new indexing algorithms, comparison and classification methods is part of our future work.

#### ACKNOWLEDGMENT

This work is being supported by a grant from “Research Projects and Technology Innovation Support Program (Programa de Apoyo a Proyectos de Investigación e Innovación Tecnológica, PAPIIT”, UNAM Project IN114413 named “Universal Evaluation System Data Quality (Sistema Evaluador Universal de Calidad de Datos)”.

#### REFERENCES

- [1] P. Christen, *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*, Series Data-Centric Systems and Applications, Springer, 2012.
- [2] E. Rahm and H.H. Do, “Data cleaning: Problems and current approaches”. *IEEE Data Engineering Bulletin* 23(4), 2000, pp.3-13.
- [3] P. Angeles and F. Garcia-Ugalde, “A Data Quality Practical Approach”, para el “International Journal On Advances in Software” Vol. 2, No. 3, 2009, pp. 259-274.
- [4] J. Bleiholder and F. Naumann, “Data fusion”, *ACM Computing Surveys* 41(1), 2008, pp. 1-41.
- [5] Febrl – A Freely Available Record Linkage System with a Graphical User Interface, *Proceeding of the 2nd Australasian Workshop on Health Data and Knowledge Management (HDKM)*, Wollongong, Australia, 2008, pp.17-25.
- [6] F. Naumann, J. Freytag, and U. Lesser, “Completeness of Integrated Information Sources”, *Workshop on Data Quality in Cooperative Information Systems (DQCIS2004)*, Cambridge, Mass., 2004, pp.583-615.
- [7] P. Angeles and F. Garcia-Ugalde, “Assessing data quality of integrated data by quality aggregation of its ancestors”, *Computación y Sistemas*, Centro de Investigación en Computación, Instituto Politécnico Nacional (IPN), vol. 13 No. 3, 2010, pp. 331-334, ISSN 1405-5546.
- [8] M. Scannapieco and C. Batini, “Completeness in the Relational Model: A Comprehensive Framework”, *Research Paper*, in *Proceedings of the 9th International Conference on Information Quality (ICIQ-04)*, Cambridge, MA, USA, 2004, pp. 333-354.
- [9] T. Churches, P. Christen, K. Lim, and J. X. Zhu, *Preparation of name and address data for record linkage using hidden Markov models*. *BioMed Central Medical Informatics and Decision Making* 2(9), 2002.
- [10] M. Odell and R. Russell, *The soundex coding system*. US Patents 1261167, 1918.
- [11] C. L. Borgman and S. L. Siegfried, “Getty’s synonyme™ and its cousins: A survey of applications of personal name-matching algorithms”, *Journal of the American Society for Information Science* 43(7), 1992, pp. 459–476.
- [12] L. Philips, “The double metaphone search algorithm”, *C/C++ Users J.* 18, 6, pp. 38-43, 2000.
- [13] M. A. Jaro, “Advances in record-linkage methodology applied to matching the 1985 Census of Tampa, Florida”, *Journal of the American Statistical Association* 84, 1989, pp. 414–420.
- [14] W. Winkler, “String comparator metrics and enhanced decision rules in the Fellegi-Sunter model of record linkage”, *Proceedings of the Section on Survey Research Methods*, 1990, pp. 354–359, American Statistical Association.
- [15] M. Neiling and H. J. Lenz, “Supplement of Information: Data Integration by Classification of Pairs of Records, Classification, Automation, and New Media Studies in Classification, Data Analysis, and Knowledge Organization”, 2002, pp. 219-226, [http://dx.doi.org/10.1007/978-3-642-55991-4\\_23](http://dx.doi.org/10.1007/978-3-642-55991-4_23) [retrieved: february, 2014], Springer Berlin Heidelberg, ISBN 978-3-540-43233-3.
- [16] I. P. Fellegi and A. B. Sunter, “A theory for record linkage”, *Journal of the American Statistical Association* 64(328), 1969, pp. 1183–1210.
- [17] D. Barone, A. Maurino, F. Stella, and C. Batini, “A privacy-preserving framework for accuracy and completeness quality assessment”, *Emerging Paradigms in Informatics, Systems and Communication*, 2009, p. 83.
- [18] P. Christen and K. Goiser, “Quality and complexity measures for data linkage and deduplication,” In: F. Guillet, H. Hamilton (eds.) *Quality Measures in Data Mining, Studies in Computational Intelligence*, vol. 43, 2007, pp. 127–151, Springer.
- [19] I. H. Witten, A. Moffat, and T. C. Bell, *Managing Gigabytes*, 2 Ed. Morgan Kaufmann, 1999.

# A Semi-automatic Method to Fuzzy-Ontology Design by using Clustering and Formal Concept Analysis

Amira Aloui  
 Université Tunis El Manar  
 Ecole Nationale d'Ingénieurs de  
 Tunis  
 LR-SITI  
 Tunis, Tunisia  
 aloui\_amira@yahoo.fr

Alaa Ayadi  
 Université Tunis El Manar  
 Ecole Nationale d'Ingénieurs de  
 Tunis  
 LR-SITI  
 Tunis, Tunisia  
 alaa.ayadi@gmail.com

Amel Grissa-Touzi  
 Université Tunis El Manar  
 Ecole Nationale d'Ingénieurs de  
 Tunis,  
 Faculté des Sciences de Tunis,  
 LIPAH  
 Tunis, Tunisia  
 amel.touzi@enit.rnu.tn

**Abstract**—Ontology design is a complex and time-consuming process. It is extremely difficult for experts to discover ontology from given data or texts. This paper presents a semi-automatic method for Fuzzy Ontology extraction and Design (FOD). The method is based on conceptual clustering, fuzzy logic and Formal Concept Analysis (FCA). The FOD approach starts with the organization of the data in homogeneous clusters having common properties which allows to deduce the data's semantic. Then, it models these clusters by an extension of the FCA. This lattice will be used to build a core of ontology that is represented as a set of fuzzy rules. Ontology designer is given this initial ontology expression for further extension by adding concepts and relationships (part-of, related to, etc.). To validate our approach, we used Protégé 4.3, that support the fuzzy concept and generates automatically the script in fuzzy-OWL 2 language.

**Keywords**—Data Mining; Clustering; Formal Concept Analysis; Fuzzy Logic; Ontology; Fuzzy OWL2.

## I. INTRODUCTION

Manual construction and description of field-specific ontology is a complex and time-consuming procedure. The recent study of ontology design methodologies shows that it is very difficult for a designer to create a precise and consistent ontology [1]. Many researchers in the field of data mining have tried to build an ontology for data mining that intended to solve some specific problems. Most of the developments aimed to automate the planning of data mining workflows [2][3]. Some of them are concerned with the description of the data mining services on the grid [4]. Others explored the possible interactions among FCA and Ontology in the Semantic Web [5] and the text documents [6] fields. The problem of these ontologies is that they are not constructed to describe the complete domain of data mining, but are simply made with a specific task in mind. Accordingly, the limits of these approaches reside in the extraction of this ontology starting from the data or a data variety, which may be huge. The goal of this paper is to present a new semi-automatic approach to extract ontology using clustering and FCA combined with a fuzzy rule-based language[19].

Our approach provides tools for semi-automatic extraction of taxonomy and automatic transformation of initial ontology

to fuzzy rules. Validation of ontology is done by using Protégé 4.3 [15].

Thus, we propose a new approach for generating an ontology which takes into consideration another degree of granularity in the process of this generation. Indeed, we propose to define an ontology between classes resulting from a preliminary classification of the data and not from the initial large amount of data. We have proven that this approach optimizes the definition of the ontology, offers a better interpretation of the data and optimizes both the space memory and the time spent on data exploiting.

The remainder of the paper is formed as follows: Section 2 introduces the basic concepts of ontology and FCA. Section 3 presents related work; Section 4 presents our motivation for this work. Section 5 describes our new approach for the semi-automatic generation of Fuzzy Ontology of Data Mining, called FODM. Section 6 validates our approach and represents some applications using the generated fuzzy ontology. Section 7 enumerates the advantages of the proposed approach. We finish this paper with a conclusion and a presentation of some future works.

## II. BASIC CONCEPTS

In this section, we present the basic concepts of ontology and FCA.

### A. Ontologies

Ontologies [7] are content theories about the classes of individuals, properties of individuals, and relations between individuals that are possible in a specified domain of knowledge. They set the terms for describing our knowledge around the field. An ontology of a domain is beneficial in establishing a common vocabulary describing the domain of interest. This is important for the unification and the sharing of knowledge about the domain and connecting with other domains. In reality, there is no common formal definition of what an ontology is. All the same, most approaches share a few core items, such as: concepts, a hierarchical IS-A-relation, and further relations. For the sake of generality, we do not discuss more specific features like constraints, functions, or axioms in this paper, instead we formalize the core in the following way:

**Definition:** A (core) ontology is a tuple  $O = (C, is\_a, R, \sigma)$  where

- $C$  is a set, whose elements are called *concepts*
- $\text{is\_a}$  is a partial order on  $C$  (I. e.,  $\text{is\_a}$  is a binary relation  $\text{is\_a}(C, X C$  which is reflexive, transitive, and anti symmetric),
- $R$  is a set whose elements is called *relation names* (or *relations* for short),
- $\sigma : R \rightarrow C^+$  is a function which assigns to each relation name its arity.

In the last years, several languages have been developed to describe ontologies. For instance, the Ontology Web Language (OWL) [8] and extension of OWL language like OWL 2 [9] or Fuzzy OWL [10]. Likewise, the number of environments and tools for building ontologies has grown exponentially. These tools aimed to provide support for the ontology's development process and for the subsequent ontology usage. Among these tools, the most relevant are: Ontolingua [11], WebODE [12], Protégé-2000 [13], OntoEdit [14] and OilEd [15].

### B. Formal concept analysis (FCA)

FCA is a method of data analysis, knowledge representation and information management. It was suggested by Rudolf Wille in 1982 [16]. In late years, FCA has grown into an international research community with applications in many fields, such as linguistics, software technology, psychology, medicine, AI, database, library science, environmental science, information retrieval, ontology building, etc. FCA starts with the concept of a formal context specifying which objects have attributes and thus a formal context may be viewed as a binary relation between the object set and the attribute set. In [17], an ordered lattice extension theory has been proposed: Fuzzy Formal Concept Analysis (FFCA), in which uncertainty information is directly represented by a real number of membership values in the range of  $[0,1]$ , then the intersection of these membership values should be the minimum of these membership values, according to fuzzy theory [18]. This number is equal to the similarity defined as follows:

**Definition.** The similarity of a fuzzy formal concept  $C_1 = (\varphi(A_1), B_1)$  and its subconcept  $C_2 = (\varphi(A_2), B_2)$  is defined as:

$$S(C_1, C_2) = \frac{|\varphi(A_1) \cap \varphi(A_2)|}{|\varphi(A_1) \cup \varphi(A_2)|} \quad (1)$$

In (1),  $\cap$  and  $\cup$  refer to the intersection and union operators on fuzzy sets [18], respectively. In [19], we showed that these FFCA are very powerful in the interpretation of the results of the Fuzzy Clustering as well as in the optimization of the flexible query.

### III. RELATED WORK

Usually, the ontology building is performed manually, but researchers try to build an ontology automatically or semi automatically to save the time and the efforts of building the ontology. We survey in this section the most important approaches that generate ontologies from data.

Clerkin et al. used concept clustering algorithm (COBWEB) to automatically discover and generate ontology. They argued that such an approach is highly appropriate to domains where no expert knowledge exists, and they proposed how they might use software agents to collaborate, as a substitute to human beings, in the construction of shared ontologies [20]. Blaschke et al. presented a methodology that creates structured knowledge for gene-product function directly from the literature. They apply an iterative statistical information extraction method combined with the nearest neighbor clustering to create an ontology structure [21]. FCA is an efficient technique that can formally abstract data as conceptual structures [22]. Quan et al. proposed to incorporate fuzzy logic into FCA to enable FCA to deal with uncertainty in data and interpret the concept hierarchy reasonably, the proposed framework is known as FFCA. They use FFCA for automatic generation of ontology for scholarly Semantic Web [23]. Dahab et al. presented a framework for constructing ontology from natural English text namely TextOntEx. TextOntEx constructs ontology from natural domain text using semantic pattern-based approach, and analyzes natural domain text to extract candidate relations, then maps them into a meaning representation to facilitate ontology representation [24]. Wuermli et al. used different ways to build ontologies automatically, based on data mining outputs represented by rule sets or decision trees. They used the semantic web languages, RDF, RDF-S and DAML+OIL for defining ontologies [25].

### IV. MOTIVATION

The motivation for developing an ontology of data mining is multi-fold.

- The area of data mining is rapidly developing and one of the most challenging problems deals with developing a general framework for data mining. By developing an ontology of data mining, we are taking one step towards solving this problem.
- There exist several proposals for ontology of data mining, but all of them are light-weight, aimed at covering a particular use-case in data mining and are of a limited scope and highly use-case dependent.

Accordingly, we would argue that the limits of these approaches are due to the extraction of this ontology departing from the data or a data variety, which may be huge. To solve all these problems, we propose a new approach for generation of the ontology using conceptual clustering, fuzzy logic, and FCA. Indeed, we propose to define an ontology between classes resulting from a preliminary classification of the data. The data classification is to divide a data set into subsets, called classes, so that all data in the same class are similar and data from different classes are dissimilar.

### V. PRESENTATION OF THE FUZZY ONTOLOGY DESIGN: FOD

#### A. Principle of the FOD

In this section, we present the architecture of the Fuzzy Ontology Design (FOD) approach and the process for building fuzzy ontology. Our FOD approach takes the

database records and provides the corresponding Fuzzy Ontology Design; Figure 1 shows the proposed approach.

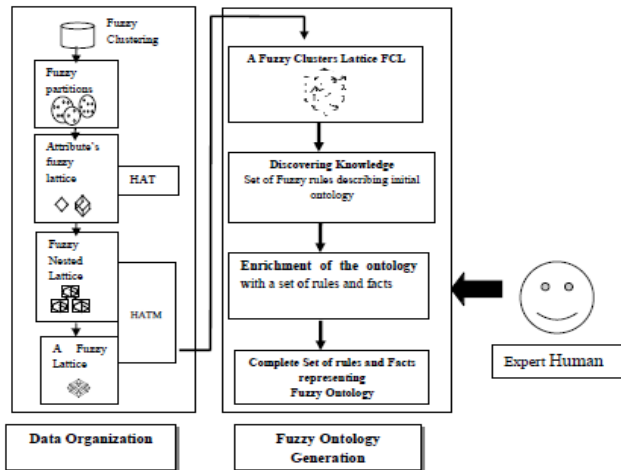


Figure 1. Presentation of the Fuzzy Ontology of Data Mining approach

The FODM approach is organized according to two principal steps: Data Organization step and Fuzzy Ontology Generation step. In the following we describe each step of the method in more detail.

*B. Theoretical Foundation of the FOD model*

In this part, we present the theoretical foundations of the proposed approach, based on the following properties:

**Property 1.** The number of clusters generated by a clustering algorithm is always lower than the number of starting objects to which one applies the clustering algorithm.

- All objects belonging to one same cluster have the same proprieties. These characteristics can be easily deduced knowing the center and the distance from the cluster.
- The size of the lattice modeling the properties of the clusters is lower than the size of the lattice modeling the properties of the objects.
- The management of the lattice modeling the properties of the clusters is optimum than the management of the lattice modeling the properties of the objects.

**Property 2.** Let C1, C2 be two clusters, generated by a clustering algorithm and verifying the properties p1 and p2 respectively. Then the following properties are equivalent:

$$C1 \Rightarrow C2 \text{ (CR)}$$

⇔

- $\forall$  object O1  $\in$  C1  $\Rightarrow$  O1  $\in$  C2 (CR)
- $\forall$  object O1  $\in$  C1, O1 checks the property p1 of C1 and the property p2 of C2. (CR)

**Property 3.** Let C1, C2 and C3 are three clusters generated by a classification algorithm and verifying the properties p1, p2 and p3 respectively. Then the following properties are equivalent: C1, C2  $\Rightarrow$  C3 (CR)

⇔

- $\forall$  object O1  $\in$  C1  $\cap$  C2  $\Rightarrow$  O1  $\in$  C3 (CR)
- $\forall$  object O1  $\in$  C1  $\cap$  C2 then O1 checks the properties p1, p2 and p3 with (CR).

The validation of the two properties rises owing to the fact that all objects which belong to a same cluster check necessarily the same attribute as their cluster.

*C. Data Organization Step*

This step allows us to organize the database records in homogeneous clusters having common properties. This step gives a certain number of clusters for each attribute. Each tuple has values in the interval [0,1] representing these membership degrees according to the formed clusters. We propose to leave the fuzzy formal context, to apply an  $\alpha$ -Cut (2) to the set of the degrees of membership, to replace them by values 1 and 0 and to deduce the binary reduced formal context. We define  $\alpha$ -Cut as follow:

**Definition.** *alpha-cut* We define the cut, noted  $\alpha$ -Cut, on the fuzzy context as being the reverse of the number of clusters obtained.

$$\alpha\text{-Cut} = (c)^{-1} \quad (2)$$

Linguistic labels, which are fuzzy partitions, will be assigned to the attribute's domain. This step consists of generating the relieving attributes for the fuzzy concept [19] lattices noted as TAH's and the fuzzy nested lattice noted as MTAH's. This step is very important in the FOD process because it allows us to define and interpret the distribution of objects in the various clusters.

**Example:** Let a relational database table presented in Table I containing the list of AGE and SALARY of Employee.

TABLE I. A RELATIONAL DATABASE TABLE.

	SALARY	AGE
t1	800	30
t2	600	35
t3	400	26
t4	900	40
t5	1000	27
t6	500	30

TABLE II. FUZZY CONCEPTUAL SCALES FOR AGE AND SALARY ATTRIBUTES

	SALARY			AGE	
	C1	C2	C3	C4	C5
t1	0.1	0.5	0.4	0.5	0.5
t2	0.3	0.6	0.1	0.4	0.6
t3	0.7	0.2	0.1	0.7	0.3
t4	0.1	0.4	0.5	0.2	0.8
t5	-	0.5	0.5	0.6	0.4
t6	0.5	0.5	-	0.5	0.5

Table II shows the results of fuzzy clustering (using Fuzzy C-Means [26]) applied to Age and Salary attributes. For Salary attribute, fuzzy clustering generates three clusters (C1, C2 and C3). For AGE attribute, two clusters have been generated (C4 and C5).

In our example,  $\alpha$ -Cut (Salary) = 0.3 and  $\alpha$ -Cut (Age) = 0.5; so, the Table II can be rewritten, as show in Table III.

TABLE III. FUZZY CONCEPTUAL SCALES FOR AGE AND SALARY ATTRIBUTES WITH  $\alpha-Cut$ .

	SALARY			AGE	
	C1	C2	C3	C4	C5
t1	-	0.5	0.4	0.5	0.5
t2	0.3	0.6	-	-	0.6
t3	0.7	-	-	0.7	-
t4	-	0.4	0.5	-	0.8
t5	-	0.5	0.5	0.6	-
t6	0.5	0.5	-	0.5	0.5

The minimum value (maximal, respectively) of each cluster corresponds to the lower (resp. higher) interval terminal of its values. The corresponding SALARY TAH of fuzzy context presented in Table III are given by the line diagrams presented in Figure 2.

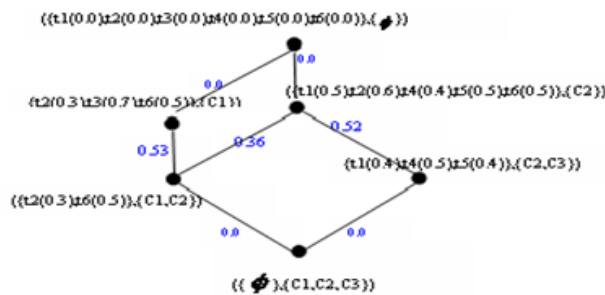


Figure 2. Salary TAH

Each cluster of a partition is labeled with a *linguistic label* provided by the user or a domain expert. For example, the fuzzy labels *young* and *adult* could belong to a partition built over the domain of the attribute *AGE*. Besides, the fuzzy labels *low*, *Medium* and *High*, could belong to a partition built over the sphere of the attribute *Salary*. Table IV presents the correspondence of the linguistic labels and their designations for the attributes *Salary* and *Age*. The corresponding fuzzy concept lattices of fuzzy context are shown in Table V.

TABLE IV. CORRESPONDENCE OF THE LINGUISTIC LABELS AND THEIR DESIGNATIONS

Attribute	Linguistic labels	Designation
Salary	Low	C1
Salary	Medium	C2
Salary	High	C3
Age	Young	C4
Age	Adult	C5

TABLE V. FUZZY CONCEPTUAL SCALES FOR AGE AND SALARY ATTRIBUTES WITH  $\alpha-Cut$ .

	SALARY			AGE	
	Low C1	Medium C2	High C3	Young C4	Adult C5
t1	-	0.5	0.4	0.5	0.5
t2	0.3	0.6	-	-	0.6
t3	0.7	-	-	0.7	-
t4	-	0.4	0.5	-	0.8
t5	-	0.5	0.5	0.6	-
t6	0.5	0.5	-	0.5	0.5

This very simple sorting procedure gives us for each many-valued attribute the distribution of the objects in the line diagram of the chosen fuzzy scale. Usually, we are interested in the interaction between two or more fuzzy many-valued attributes. This interaction can be visualized using the so-called fuzzy nested line diagrams. It is used for visualizing larger fuzzy concept lattices, and combining fuzzy conceptual scales on-line. Figure 3 shows the fuzzy nested lattice constructed from TAH's.

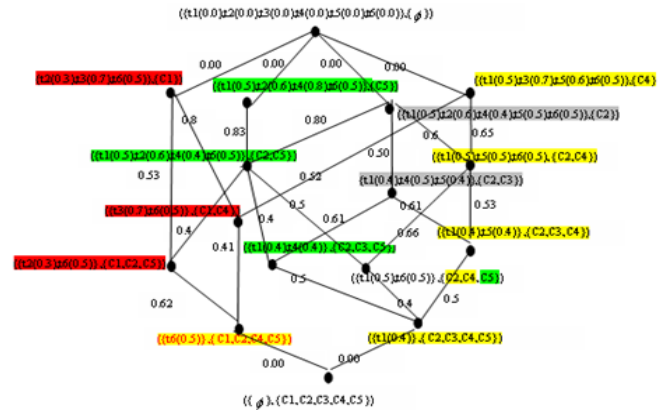


Figure 3. Fuzzy Lattice: MTAH

D. Fuzzy Ontology Generation step

This step consists of the construction of a Fuzzy Ontology from the Fuzzy Cluster Lattice generated in the first step.

1) FCL Generation.

The goal of this phase is to make a certain abstraction on the list of the objects with their degrees of membership in the clusters. This lattice will be used to build a core of ontology.

**Definition.** A Fuzzy Clusters Lattice (FCL) of a Fuzzy Formal Concept Lattice, consists on a Fuzzy concept lattice where each equivalence class (a node of the lattice) contains only the intentional description (intent) of the associated fuzzy formal concept.

**Definition.** A level *i* of FCL is a set of nodes of FCL with cardinality equal to *i*.

We do a certain abstraction of the list of the objects with their degrees of membership in the clusters. The nodes of FCL are the clusters ordered by the inclusion relation.

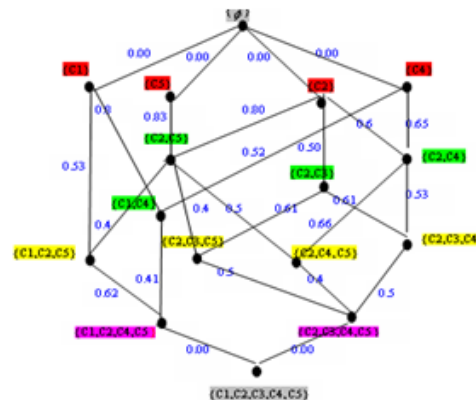


Figure 4. Fuzzy Clusters Lattice FCL

As shown in Figure 5, we obtain a lattice more reduced, simply traversed and stored.

2) *Discovering Knowledge*

This step consists in the Extraction of knowledge for the fuzzy ontology. To do so, we must build a concept of hierarchy from the conceptual clusters; we need to find the hierarchical relations from the clusters. We first define a concept hierarchy as follows:

**Definition** (Concept Hierarchy). A concept hierarchy is a poset (partially ordered set)  $(H, <)$ , where  $H$  is a finite set of concepts and  $<$  is a partial order on  $H$ .

a) Principle of discovering knowledge from FCL.

Taking as input an FCL, the extraction of fuzzy association rules can be performed straightforwardly. Indeed, the rule represents the implications deduced from FCL between two adjacent classes. The confidence factor will be equal to the weight of the link (arc) between the two nodes.

**Rule 1. Discovering rule:** Let  $C1 = \{A1, \dots, A\}$  and  $C2 = \{B1, \dots, Bm\}$  two nodes of FCL such as  $C2$  is the successors of  $C1$  in the lattice and having as distance  $d > 0$  (weight of the arc) the generated rule will be defined as follows:

$$A1, \dots, An \Rightarrow B1, \dots, Bm \quad (d)$$

Notice that, if  $d=0$  this implies that there is no object in common to the two concepts  $C1, C2$ . There is no knowledge to discover or to generate.

**Rule 2. Discovering rule:** Let  $C1 = \{A1, \dots, An\}$  and  $C2 = \{B1, \dots, Bm\}$  two nodes of FCL such as  $C2$  is the successors of  $C1$  in the lattice and having as distance  $d > 0$  (weight of the arc). The generated rule will be defined by:

$$R: A1, \dots, An \Rightarrow C1, \dots, Cq \quad (d) \text{ such that } \\ \{C1, \dots, Cq\} = \{B1, \dots, Bm\} \setminus \{A1, \dots, A\} \quad (\forall Ci, Ci \in \{A1, \dots, An\})$$

**Rule 3. Generated rule:** Let  $C1 = \{A1, \dots, An\}$   $C2 = \{B1, \dots, Bn\}$  and  $C3 = \{D1, \dots, Dn\}$  three concepts such as  $C2$  successors of  $C1$  and  $C3$  successor of  $C2$  having respectively as distance  $d1$  and  $d2$ . The generated rule will be defined by:

$$R3: A1, \dots, An \Rightarrow D1, \dots, Dn \quad (d2 * d1)$$

b) *Algorithm for Discovering Fuzzy Association rules.*

The pseudo-code for this algorithm is as follows:

```

Generating knowledge
Input: Fuzzy Cluster Lattice FCL
Output : FRS: Fuzzy Rules Set
Begin
FRS := ∅
Find_Cluster(FCL, 0, S) ;
For each subconcept Cj = {y1, ..., ym} of
Ci in S
    r.premise = ∅
    r.conclusion = {y1, y2, ..., ym}
    r.CF = 1
    FRS = FRS ∪ {r}
End For
Nmax := Niveau_max(FCL) ;
For Niv := 1 to Nmax-1 do
    Find_Cluster(FCL, i, S) ;
    For Ci = {x1, ..., xm} in S do
        For each subconcept Cj = {y1, ..., ym} of
        Ci and having (d > 0)
            r.premise = {x1, x2, ..., xn}
            r.conclusion = {y1, y2, ..., ym} \ {x1, x2, ..., xn}
            r.CF = d
            FRS = FRS ∪ {r}
        End For
    End For
End For
Generate_Rule(FRS, FRS1);
FRS := FRS ∪ FRS1
End.
    
```

Figure 5. Algorithm for Discovering Fuzzy Association rules

The Algorithm for Discovering Fuzzy Association rules traverses the search space (FCL) by level to square up the Fuzzy Rules Set (FRS). As input it takes the lattice of Clusters FCL and returns, as output, the list of all Fuzzy Rules Set (FRS) generated. It works as follows: For each non empty node  $\in$  FCL in descending, it generates all rules with one cluster in conclusion (level 1). Then, it generates the set of all rules with two Clusters in conclusion. The same process is applied to generate conclusions with four clusters, and so on until conclusions with  $n$  clusters are generated.

**Proposition 3.**

If the system of extraction rules traverses the search space by the level of the lattice of clusters then no rule generated by this system is redundant (all the generated rules are obligatorily distinct).

**Proof.** This is due to the fact that from a level to another of the lattice the nodes are obligatorily distinct (by definition even of a level of lattice).

3) *Ontology Generation.*

This step constructs fuzzy ontology from a fuzzy context using the concept hierarchy created by fuzzy conceptual clustering. This is done based on the characteristic that both FCA and ontology support formal definitions of concepts. Thus, we define the fuzzy ontology as follows:

**Definition (Fuzzy Ontology).** A fuzzy ontology  $F_o$  consists of four elements  $(C, A^C, R, X)$ , where:

- $C$  represents a set of concepts,
- $A^C$  represents a collection of attribute sets, one for each concept,
- $R = (R_T, R_N)$  represents a set of relationships, which consists of two elements:
  - $R_N$  is a set of non-taxonomy relationships and
  - $R_T$  is a set of taxonomic relationships.
- Each concept  $c_i$  in  $C$  represents a set of objects, or instances, of the same kind.
- Each object  $o_{ij}$  of a concept  $c_i$  can be described by a set of attributes values denoted by  $A^C(c_i)$ .
- Each relationship  $r_i(c_p, c_q, \alpha)$  in  $R$  represents a *fuzzy association* between concepts  $c_p$  and  $c_q$ , and the instances of such a relationship are pairs of  $(c_p, c_q)$  concept objects with confidence  $\alpha$ ;  $\alpha$  is in  $]0..1]$ .
- Each attribute value of an object or the relationship instance is associated with a fuzzy membership value between  $[0,1]$  implying the uncertain degree of this attribute value or relationship.
- $X$  is a set of axioms. Each axiom in  $X$  is a constraint on the concept's and relationship's attribute values or a constraint on the relationships between concept objects.

In our approach, we consider the Fuzzy Ontology Lattice as a formal domain-specific ontology. This ontology has all lattice properties, which are useful for ontology sharing, and reasoning. The whole process to create a fuzzy ontology was completed. We may consider nodes as concepts. The name of the concept is a concatenation of an attribute and its label linguistics, in accordance with the correspondence in Table IV. Nevertheless, taxonomic relationships between concepts are present in the lattice.



### VI. VALIDATION AND APPLICATION OF GENERATING FUZZY ONTOLOGY

The performance of the proposed algorithm for Discovering Fuzzy Association rules can be measured in order to evaluate the generated ontology. To do this, we evaluate the processing time and the number of rules between two approaches: The first one does not apply the clustering concept and the second uses the formal concepts for structuring and building ontology-based classification.

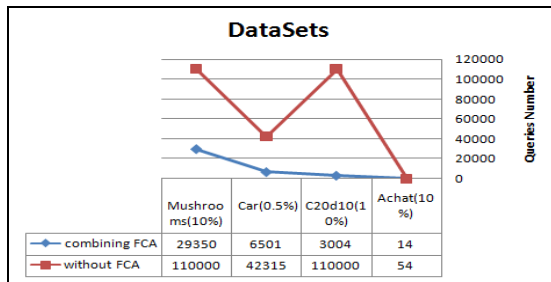


Figure 6. Metrics of the Proposed Approach

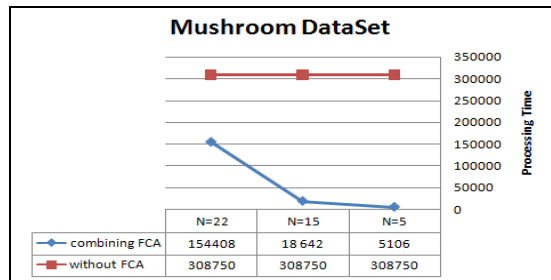


Figure 7. Processing time of the Proposed Approach

We prove that with FCA, we minimize the high time and space complexity of the resulting lattice. We implement, then, the concept lattice (result of fuzzy classification in ClusterFCA) with Protégé 4.3, generate the ontology, test its consistency, and extract the queries. The process of generating ontology is presented in Figure 9.

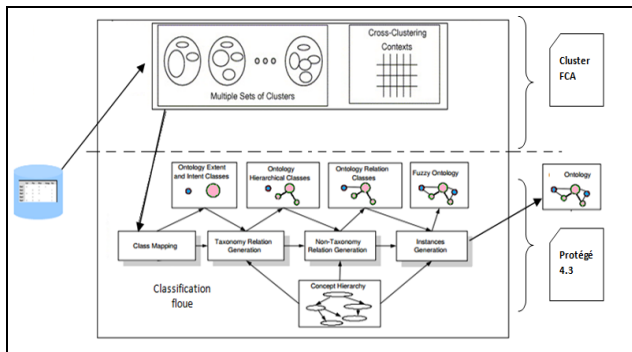


Figure 8. Process of generating/ validating Ontology

By taking the abstractions got by FCA as a guideline, the generated ontology in Protégé 4.3 is shown in Figure 10.

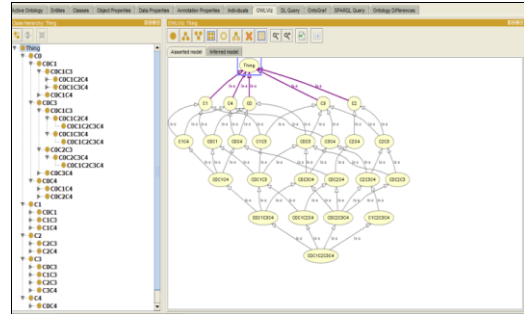


Figure 9. Generated Ontology

Once the queries concepts are defined, we can model the resulting rules deduced from our Fuzzy Ontology using Protégé 4.3 and respond to the user answers. We have also succeeded to generate the description of our ontology with fuzzy-OWL 2 language.

### VII. ADVANTAGES OF THE PROPOSED APPROACH

We present in Table VI the advantage of every basic concept used on our approach.

TABLE VI. ADVANTAGES OF THE PROPOSED APPROACH

Operationalization	Advantages	Comments
Using FCL for constructing Ontology	Redundant relation elimination	For each relation concepts, we can have two concept instances which are equivalent. The two concept instances are valid in the sense that concept with a higher membership degree is closer to the concept truth. Eliminating one of the concept relation will not reduce the information conveyed, but will reduce by half the size of the storage. In constructing an ontology we retain the fuzzy relation that has a higher membership degree. This decision strategy will choose a positive concept instance and will choose a stronger relation if the two membership values are close to each other.
	Less meaningful relation elimination	After redundant class relation is removed much potential less meaningful information intact.
	Unrelated concept relation elimination	The relation between two distinct classes cannot be established if both concept never co-occur so that their membership values will be 0. It is obvious that unrelated classes should also not be considered during the ontology creation. These concepts will be automatically excluded by applying alpha-cut as described above.
Using the domain ontology	Less number of generating classes	The number of classes generated is less than the number of objects starting on which we apply the classification algorithm. This improves the quality of the process of information retrieval by considering only a part of the ontology according to a user preference.
	Best answer to the user request	The ontology has been described in OWL2, we took advantage of the progress of this language in terms of expressiveness for greater capacity inference without using a dedicated language to express rules

Nowadays, a few proposals for ontologies of data mining using FCA exist, but all of them start from a data unit, after having done a data cleansing step and an elimination of invalid-value elements. We have come to the conclusion that this idea is very important because it models an abstraction of the data especially in the case of voluminous one.

### VIII. CONCLUSION

Motivated by the increased need for formalized representations of the domain of Data Mining, the success of using FCA and Ontology in several Computer Science fields, we presented in this paper a new approach for the semi automatic generation of Fuzzy Ontology Design (*FOD*), through the fusion of conceptual clustering, fuzzy logic, and FCA. In our approach, we proposed to generate an ontology taking into consideration another degree of granularity in the process of generation. Indeed, we suggest to define an ontology between classes resulting from a preliminary classification of the data. We prove that this approach optimizes the definition of the ontology, offers a better interpretation of the data and optimizes both the space memory and the execution time for exploiting this data. To validate our approach, we used Protégé 4.3, which supports the fuzzy concept, to model our ontology and to generate the script in fuzzy-OWL 2 language.

Knowing that the number of classes has been always lower than the number of starting data, our proposed approach intends to achieve the objectives of offering better interpretation of the data and minimizing both execution time and space memory (by reducing considerably the definition of the ontology). As future perspectives of this work, we intend to test our approach on several large datasets.

### REFERENCES

- [1] C. Tempich and R. Volz, "Towards a benchmark for Semantic Web reasoners-an analysis of the DAML ontology library," Sure Y (editor) Proceedings of Workshop of Evaluation of Ontology-based Tools (EON 2003) at 2nd Int. Semantic Web Conference (ISWC 2003), USA, (2003).
- [2] A. Bernstein, F. Provost, and S. Hill, "Toward intelligent assistance for a data mining process: An ontology-based approach for cost-sensitive classification", IEEE Trans on Knowl and Data Eng, 2005, pp. 503–518.
- [3] M. Zakova, P. Kremen, F. Zelezny, and N. Lavrac, "Planning to learn with a knowledge discovery ontology," In P. Brazdil, A. Bernstein, and L. Hunter, editors, Proceedings of the Second Planning to Learn Workshop (PlanLearn) at the ICML/COLT/UAI, 2008, pp. 29–34.
- [4] P. Brezany, I. Janciak, and A. M. Tjoa, "Data Mining with Ontologies Implementations, Findings and Frameworks," chapter Ontology-Based Construction of Grid Data Mining Workflows. IGI Global, 2007.
- [5] Q. T. Tho, S. C. Hui, A. C. M. Fong, and Cao, T.H., "Automatic fuzzy ontology generation for semantic web", Knowledge and Data Engineering, IEEE Transactions on, vol. 18, no. 6, 2006, pp. 842-856.
- [6] P. Cimiano, A. Hotho, G. Stumme, and J. Tane, "Conceptual knowledge processing with formal concept analysis and ontologies," In ICFA, 2004, pp. 189-207.
- [7] B. Chandrasekaran, J. R. Josephson, and V. R. Benjamins, "What are ontologies, and why do we need them?," IEEE Intelligent Systems, 1999, pp. 20–26.
- [8] S. Bechhofer et al. , "OWL Web Ontology Language: Reference". World Wide Web Consortium, February. 2004.
- [9] B. Cuenca-Grau, I. Horrocks, B. Motik, B. Parsia, P. F. Patel-Schneider, and U. Sattler, "OWL 2: The next step for OWL", Journal of Web Semantics, 2008, pp. 309-322.
- [10] F. Bobillo and U. Straccia, "Representing fuzzy ontologies in OWL 2," in: Proceedings of the 19th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2010), IEEE Press, 2010, pp. 2695–2700.
- [11] A. Farquhar, R. Fikes, and J. Rice, "The ontolingua server: a tool for collaborative ontology construction," In the 10th Knowledge Acquisition for Knowledge-Based Systems (KAW'96), Canada, 1996.
- [12] J. Arpirez, O. Corcho, M. Fernández-López and A. Gómez-Pérez). "WebODE , a Workbench for Ontological Engineering", In First international Conference on Knowledge Capture (K-CAP'01), Victoria, Canada ACM, 2001, pp. 6–13.
- [13] N. Noy, R. Fergerson, and M. Musen, "The knowledge model of Protégé2000 : Combining interoperability and flexibility," In R. D IENG & O.CORBY, Eds., 12th International Conference on Knowledge Engineering and Knowledge Management (EKAW'00), volume (1937) of Lecture Notes in Computer Science, Juan-les-Pins, France: Springer Verlag, pp. 17–32.
- [14] Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer, and D. Wenke, "OntoEdit: Collaborative Ontology Engineering for the Semantic Web", In I. Horrocks & J. Hendler, Eds., First International Semantic Web Conference (ISWC'02), volume (2342) of Lecture Notes in Computer Science, Chia, Sardaigne, Italie: Springer Verlag. 2002, pp. 221–235
- [15] S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens, "OilEd: a Reason-able Ontology Editor for the Semantic Web", In Joint German/Austrian conference on Artificial Intelligence (KI'01), volume (2174) of Lecture Notes in Artificial Intelligence, Vienne, Austria: Springer Verlag, 2001, pp. 396–408.
- [16] R. Wille, "Restructuring lattice theory: An approach based on hierarchies of concepts", In I. Rival (Ed.), Ordered sets, 1982, pp .445–470.
- [17] T. T. Quan, S. C. Hui, and T. H. Cao, "A Fuzzy FCA-based Approach to Conceptual Clustering for Automatic Generation of Concept Hierarchy on Uncertainty Data", Proc. of the 2004 Concept Lattices and Their Applications Workshop (CLA), pp. 1-12, 2004.
- [18] L. A. Zadeh, "Fuzzy Logic and Approximate Reasoning," Synthese, vol. 30, 1975, pp. 407-428.
- [19] A. Grissa Touzi, M. Sassi, and H. Ounelli, "An innovative contribution to flexible query through the fusion of conceptual clustering, fuzzy logic, and formal concept analysis", International Journal of Computers and Their Applications. Vol. 16, N 4, December. 2009, pp. 220-233.
- [20] P. Clerkin, P. P. Cunningham, and C. Hayes, "Ontology Discovery for the Semantic Web Using Hierarchical Clustering" , Proc. European Conf. Machine Learning (ECML) and European Conf. Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD-2001), 2001.
- [21] C. Blaschke and A. Valencia, "Automatic Ontology Construction from the Literature", Genome Informatics, vol. 13, 2002, pp. 201–213.
- [22] B. Ganter, G. Stumme, and R. Wille, "Formal Concept Analysis", Foundations and Applications. Lecture Notes in Artificial Intelligence, no.3626, Springer-Verlag. ISBN 3-540-27891-5. (Eds.) 2005.
- [23] T. T. Quan, S. C. Hui, A. C. M. Fong, and T. H. Cao, "Automatic generation of ontology for scholarly semantic Web", In: Lecture Notes in Computer Science. Vol. 3298, 2004 , pp. 726–740.
- [24] M. Y. Dahab, H. Hassan, and A. Rafea, "TextOntoEx: Automatic ontology construction from natural English text", Expert Systems with Applications (2007), doi:10.1016/j.eswa.2007.01.043.
- [25] O. Wuerml, A. Wrobel, S. C. Hui, and J. M. Joller, "Data Mining For Ontology\_Building: Semantic Web Overview", Diploma Thesis–Dep. of Computer Science\_WS2002/2003, Nanyang Technological University.
- [26] H. Sun, S. Wanga, and Q. Jiangb, "FCM-Based Model Selection Algorithms for Determining the Number of Clusters", Pattern Recognition 37, pp. 2027-2037.

# Modeling Ontology-based User Profiles from Company Knowledge

Silvia Calegari, Matteo Dominoni

Department of Informatics, Systems and Communication (DISCO)

University of Milano-Bicocca

Milan, Italy

e-mail: {calegari,dominoni}@disco.unimib.it

**Abstract**—This paper presents a model based on the ontology paradigm to represent the knowledge of users who work in companies. The model guarantees to follow specific guidelines that can be used in defining user knowledge. The goal is to improve the development of processes and applications aimed at leveraging the information stored in companies to support users during their interactions with the system by considering all the advantages of the ontology-knowledge.

This research was developed in the KBMS 2.0 project where an advanced portal has been defined with the aim to manage and search information effectively. The model to represent user knowledge has been defined by analysing the information of the KBMS 2.0 project stored in the database.

*User Profile, Ontology, Company Knowledge.*

## I. INTRODUCTION

In the last few decades, many research activities have focused on the management of the growing amount of information on the Web or in any big repository. There are several problems that can afflict this issue, ranging, for example, from architectural aspects, i.e., how to define a robust architecture for addressing issue of scalability, to personalization aspects, i.e., how to model user profiles to represent the user's interests and preferences. The former allows to help systems retrieve information faster, whereas the latter is used to filter knowledge which is not related to the user's interest needs.

In this paper, we focus on the definition of user profiles in the context of big companies, where thousands of pieces of information are stored in the repository. In this scenario, the problem is to support the user during his/her research activities, not only with the local search engine for retrieving relevant documents, but also with all the knowledge accessible in the company portal, such as news, emails, external links, etc., where the information is geographically distributed in the several locations of the company. This means that in a big company, besides the standard problems with a search engine on the Web, there are complex problems due to the several information access points of a user. In addition, in a company a user belongs to a hierarchical structure, where for each level users have a different role, meaning different access to the information. Thus, the definition of processes that exploit the knowledge

represented in the user profile assumes a key role for managing knowledge in companies.

The accurate definition of a user profile plays a central role in effective approaches to personalization: only if a user profile faithfully represents the information related to a user, can a system rely on it. Three main activities have to be considered during the building of a user profile, such as: (1) the identification of the knowledge which represents the user's interests, (2) the choice of a formal language used to represent this knowledge, and (3) a strategy for updating it. Regarding the first point, in the literature explicit and implicit approaches [1] are adopted in order to capture the user's interests and preferences. With the explicit approach the user must explicitly specify his/her preferences to the system by filling in questionnaires and/or by providing short textual descriptions. With the implicit approach, the user's preferences are automatically gathered by monitoring the user's actions, thanks to the use of click-through data analysis, log analysis, etc. The second point is on the choice of suited data structures for modeling a user profile; in the literature bags of words, vectors, graph-based representations, and some external knowledge sources (i.e., WordNet, or Open Directory Project) have been mainly used to define users' profiles. Ontologies are a recent powerful tool for knowledge definition, modeling and representation; and they allow to give a more structured and expressive knowledge representation with respect to the above mentioned approaches. In fact, they allow to enrich the expressiveness of the information represented in a profile by using formal languages like Resource Description Framework Schema (RDFS) or Ontology Web Language (OWL). The third point is the most complex as it has to guarantee a dynamic user profile, where the knowledge is always updated and related to the user's recent interests and preferences. The objective is to support the processes that exploit this fresh information to help users during their interactions with the system.

In the literature, the existing models that build user profiles based on ontologies are mainly focused on approaches either relying on data mining techniques [2] or adopting external reference knowledge [3] to capture the meaning of the user's preferences that are represented in RDFS or OWL. With the use of formal ontological languages it is possible to define user profiles, thanks to the adoption of suitable logical

language constructs, but they do not give explicit guidelines on how to use them in order to solve specific tasks. The Ontology Design Patterns (ODP) [4] are a recent solution to make ontology as reusable solutions, and they are defined as a “reusable successful solution to a recurrent modeling problem”. ODP are aimed at reducing mistakes in ontologies, detecting uncovered requirements, improving qualities of produced ontologies, etc. [5].

In this paper, for the first time in the literature we make use of the ODP solution to define the skeleton of user profiles that can be reusable in any domain where users are involved. Our objective is to represent a richer knowledge of user profiles than the existing models previously defined that mainly acquire initial information from textual information or from a set of keywords related to the user's interests. To help us in the definition of the ODP for user profiles, we considered the knowledge gained from a research project with the Enel SpA energy company. In this case study, user knowledge is obtained by analysing the database of the company where only the portion of information useful for the definition of some crucial parts of the ODP-user profile is extracted. In addition, we have completed some portions of the defined ODP-ontology by considering the basic notion of the ODP solution, where it is possible to compose pieces of knowledge by considering other ODPs previously defined. An ODP is used across ontologies to define a richer knowledge based on ontologies, whereas an ontology is used across applications. Our intent is to obtain an accurate ODP-user profile that can be used as a model for representing the user knowledge in any enterprise. The next phase of this work will consider the validation of the obtained user profile; at the moment this aspect is out of topics with respect to the goal of this paper.

The paper is organised as follows. Section II describes how the main pieces of knowledge for defining the ODP-user profile have been acquired. Section III presents the generic ODP-user profile for companies. Finally, in Section IV conclusion and future work are given.

## II. TOWARDS THE DEFINITION OF A ODP-USER PROFILE

This section gives the steps performed to elicit knowledge used to define the ODP-user profile. We started from the knowledge stored in the database of the KBMS 2.0 project (KBMS 2.0: Knowledge-Base Management System 2.0 is a research project that has involved the University of Milano-Bicocca, Italy, together with the Enel SpA energy company) [6]. The KBMS 2.0 project is aimed at providing an advanced knowledge-based portal to manage knowledge innovatively, with the objective to support people in all the phases of their searches and also in the emergence of new knowledge in the system. This project is based on the open source edition of the Liferay portal and is based on several modules (by developing new modules or by extending some existing ones) in order to satisfy the requirements of the Enel SpA stakeholders. The most important modules developed for the KBMS 2.0 portal are: a workflow to guarantee a

certified quality of the information, a personalised search engine to retrieve relevant documents for users, a grid of navigation to allow users immediate access to documents classified, thanks to the support of specific categories, a newsletter to send and filter news according to specific rules, spot-news to highlight emergent news, and my-links to personalise access to external web-links. All the information used by these modules is stored in the database. In detail, the Relational Database Management System (RDBMS) Oracle 10g version has been used for the project by defining a total of 253 tables. Most of the tables have been automatically defined by Liferay during its installation, while some of them are not used for the project such as tables on market-place, wikipedia, kaleo, etc. Other tables, instead, have been introduced to support newly defined modules for activities not defined in the standard edition of Liferay.

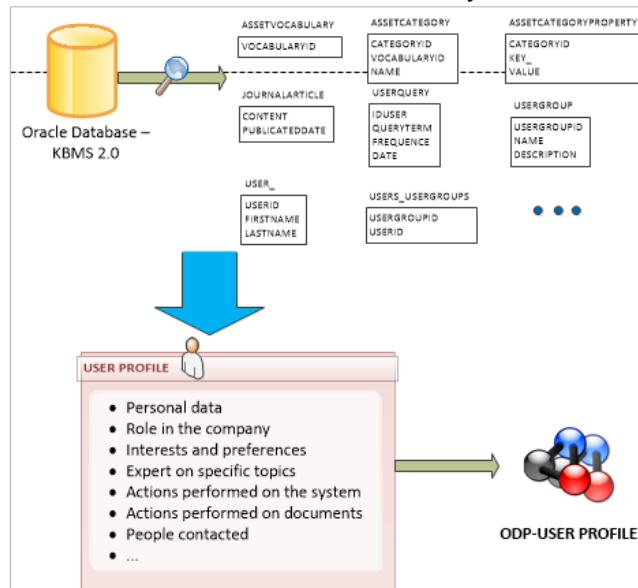


Figure 1. Logical path for the definition of a ODP-User Profile.

The database has been analysed to design for the extraction of some portions of knowledge useful in defining the skeleton of the user profile presented in Section III. Figure 1 shows the phases considered for our analysis that are: (1) view the tables stored in the database in order to consider only the subset related to the knowledge of users, (2) list the user's knowledge, and (3) define the ODP-user profile based on point (2).

The list of user knowledge assumes a key role for the definition of the user profile model, which has been grouped in the following macro-areas:

- **Personal data:** describes personal user characteristics, such as name, address and age.
- **User's company data:** describes user characteristics of the company, such as login, password, nickname and job/role.

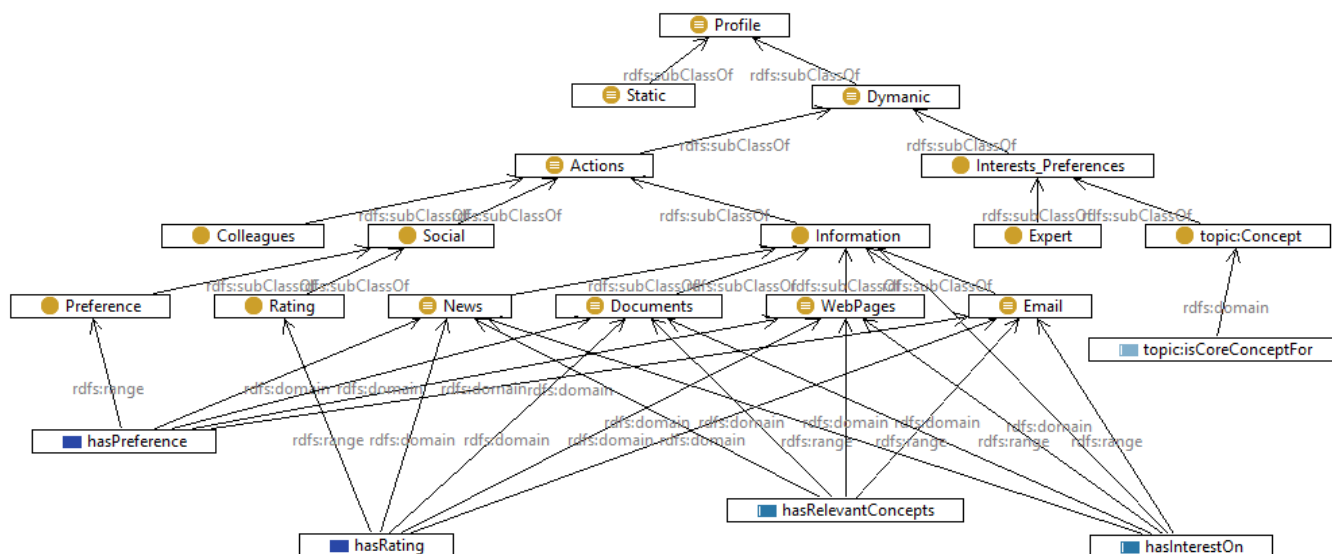


Figure 2. Example of a TWO-COLUMN figure caption: (a) this is the format for referencing parts of a figure.

- **Company data:** describes company information, such as name and address.
- **Interests and preferences:** describes user interests and preferences on topical arguments. They can be obtained by analysing, for example, the documents saved or viewed by the user.
- **Expert on specific topics:** indicates which topics a user is considered as an expert. Several processes can be adopted in order to establish the user's expertise, such as methodologies of voting or use of metrics defined in the literature.
- **Actions performed on the information:** monitors the actions on the information stored in the system, such as the pages/documents viewed, the time spent on a page/document, emails sent, etc.
- **Social Actions:** monitors the user's action on the system or documents, such as a methodology of rating on documents retrieved by a search engine, or on people contacted for suggestions, directed preferences on the information such as the use of like/not-like.
- **People contacted:** monitors the people who are working in the company that have been contacted by a user, for example, by emails or by using social applications.

The knowledge considered from these macro-areas can be gathered by using both explicit and implicit approaches, as explained in Section I. In this phase, it is not important to understand the complex processes that exploit the data stored in the log files, generally used to monitor the user's actions (i.e., actions on information and social actions), in order to understand what information can be of interest to a user. Our objective is to analyse such preferred information to design the ODP model used for establishing the knowledge that has to be represented in a generic user profile (see Section III).

### III. ODP-USER PROFILE FOR COMPANIES

The ODPs are used in the community to define reusable logical models with the aim to design conceptualizations of knowledge. In this section, we present the first ODP model defined to represent the knowledge of users who work in companies. Figure 2 shows the class diagram of the user profile-ODP where the main concepts are reported. The main concepts are organized by the taxonomic relation IS-A, whereas the other knowledge is represented by different relations (i.e., *ObjectProperties*) as described in this section. For the sake of readability, we have not introduced the pre-defined *Instances* (e.g., *male* and *female* instances for the concept *Gender*), and the relations (domain and range) established with the *DatatypeProperties* and the *ObjectProperties*.

The user profile is logically divided into two logical areas that are: *Static* and *Dynamic* with the intent of representing the two key aspects of the user's profile knowledge. The *Static* class represents long-term interests of user knowledge. The *Dynamic* class represents short-term interests of user knowledge, where the changes can happen every time that a user interacts with the system. This class identifies the most interesting part of the ODP-user profile, since it needs to represent the interactions of a user with the information stored in the system and the people who work in the company. Thus, we have classified the information into four categories that are: *Documents*, *Webpages*, *Email*, and *News*. For these categories the following common *ObjectProperties* have been defined:

*hasInterestedOn*, *hasPreference*, *hasRating*, and *hasRelevantConcepts:topicConcept*.

The *hasInterestedOn* defines user interests on specific information stored in the system, the *hasPreference* defines

the social actions on the information related to the *like/not-like* preferences, and the *hasRating* defines the social actions on the information related to the rating preferences given, for example, by the *star rating* method.

The *hasRelevantConcepts:topicConcept* deserves a closer look, as it links the documents or email or news or webpages with the relevant concepts extracted from them. The relevant concepts are not defined, as bags of words or vectors, etc., as they are in many works presented in the literature (see Section I). In our work, the relevant concepts are represented as ontology-based knowledge, not extracted by external reference knowledge (i.e., Yet Another General Ontology (YAGO) or Open Directory Project or domain ontologies). To define the knowledge for representing the relevant concepts of user preference information, we have used the ODP schema on *Topic* [7] download from the official ODP's repository [4] with the intent of following the logic of the ODP models, i.e., to reuse pre-defined logical models in new ODP models.

In the end, the class *Colleagues* defines the users who work in the same company that have interacted with the user. The idea is to create a network of users to establish, for example, people who share common interests.

#### A. How to use the ODP-User Profile

When the ODP-User Profile is defined, it is possible to use it in order to represent the knowledge of a user profile represented as an ontology. The next phase consists in modeling an ontology in the OWL language that is an instance of the ODP-User Profile. This step is easily performed by importing the ODP schema in the configuration part of the OWL file as follows:

```
<?xml version="1.0"?>
<rdf:RDF
...
  xmlns="http://www.ontologydesignpatterns.org/cp/
  examples/userprofile/enel.owl#"
  xmlns:up="http://www.ontologydesignpatterns.org/cp/
  owl/userprofile.owl#"
  xml:base="http://www.ontologydesignpatterns.org/cp/
  examples/agentrole/enel.owl">
  <owl:Ontology rdf:about="">
  ...
  <rdfs:comment>It encodes the following:User Profile
  in the Enel SpA company</rdfs:comment>
  <owl:imports rdf:resource=
  "http://www.ontologydesignpatterns.org/cp/owl/
  userprofile.owl"/>
  </owl:Ontology>
  <up:Profile>
  ...
  </up:Profile>
</rdf:RDF>
```

## IV. CONCLUSION AND FUTURE WORK

In this paper, we have presented the ODP model for representing the user profile logical schema. An ODP model is a new frontier of research, as it allows to provide specific guidelines for defining knowledge based on specific logical schemas. In fact, the defined ODP-user profile can be used to represent the knowledge of users in companies to facilitate the definition of applications and of processes by exploiting all the well known advantages of the ontology paradigm. The ODP-user profile has been designed by taking support from the knowledge stored in the database of the KBMS 2.0 project. In this project, an advanced portal has been defined with the aim of satisfying the user's information needs by providing a certified quality of information and by supporting a user during his/her searches, with the goal of retrieving relevant documents faster.

In the future, we plan to improve the ODP-user profile by including more details on the defined concepts and by defining axioms to classify user interests on information and on people to support the processes that are defined for this goal. Furthermore, we will consider its validation in order to establish the good quality of the obtained ODP-user profile.

#### ACKNOWLEDGMENT

The authors would like to thank all the people who allowed us to make an in depth study for the development of the KBMS 2.0 system.

#### REFERENCES

- [1] J. Teevan, S. T. Dumais, and E. Horvitz, "Personalizing search via automated analysis of interests and activities". In Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval, ser. SIGIR '05. New York, NY, USA: ACM, 2005, pp. 449–456, doi:10.1145/1076034.1076111.
- [2] X. Tao, Y. Li, and N. Zhong, "A personalized ontology model for web information gathering". IEEE Trans. on Knowl. and Data Eng., vol. 23, April 2011, pp. 496–511, doi: 10.1109/TKDE.2010.145.
- [3] S. Calegari and G. Pasi, "Personal ontologies: Generation of user profiles based on the yago ontology". Inf. Process. Manage., vol. 49, no. 3, 2013, pp. 640–658, doi: http://dx.doi.org/10.1016/j.ipm.2012.07.010.
- [4] Ontology Design Pattern, website. [Online]. Available: [http://ontologydesignpatterns.org/wiki/Main\\_Page](http://ontologydesignpatterns.org/wiki/Main_Page) (October, 2013)
- [5] V. Presutti, E. Blomqvist, E. Daga, and A. Gangemi, "Pattern-based ontology design". In Ontology Engineering in a Networked World, M. del Carmen Suarez-Figueroa, A. Gomez-Perez, E. Motta, and A. Gangemi, Eds. Springer, 2012, pp. 35–64, doi:10.1007/978-3-642-24794-1\_3.
- [6] S. Calegari, M. Dominoni, and E. Panzeri, "Towards the design of an advanced knowledge-based portal for enterprises: the KBMS 2.0 project". The 27th International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems, 2013, Unpublished.
- [7] Ontology Design Pattern, Topic schema, website. [Online]. Available: <http://www.ontologydesignpatterns.org/cp/owl/topi.c.owl> (October, 2013)

## A Concept for Plagiarism Detection Based on Compressed Bitmaps

Andreas Schmidt<sup>\*†</sup>, Reinhold Becker<sup>‡</sup>, Daniel Kimmig<sup>†</sup>, Robert Senger<sup>\*</sup> and Steffen Scholz<sup>†</sup>

<sup>\*</sup> Department of Computer Science and Business Information Systems,  
Karlsruhe University of Applied Sciences  
Karlsruhe, Germany

Email: {andreas.schmidt, robert.senger}@hs-karlsruhe.de

<sup>†</sup> Institute for Applied Computer Science  
Karlsruhe Institute of Technology  
Karlsruhe, Germany

Email: {andreas.schmidt, daniel.kimmig, steffen.scholz}@kit.edu

<sup>‡</sup> esentri AG, Ettlingen

Email: reinhold.becker@esentri.com

**Abstract**—In the last few years, several public persons in Germany have been convicted of inadequate scientific practice and or scientific misconduct in writing their dissertations. An examples is the former German Federal Minister of Defense Guttenberg, who had to resign as a consequence of this scandal. These events have led to an increased interest in methodologies to automatically detect plagiarism in documents. In today’s digital society, however, the vast growth of available information makes this a challenging task. To address this situation, tools for the detection of plagiarism must be built up on highly efficient data structures and utilize very fast operations. In our approach, we propose the use of compressed bitmaps as a representation form. We introduce a new concept of plagiarism detection, which is based on mapping suspicious documents and potential source documents onto these compressed bitmaps. We will explain how the detection process can be accelerated.

**Keywords**—Compressed bitmaps; plagiarism detection; visualization

### I. INTRODUCTION

The fully automated search for plagiarized sections in documents is gaining more and more importance. This search is a multi-stage process [1] with the initial point being a suspicious document that has to be examined as to whether its content is plagiarized. In a first preselection step, which is called *source retrieval*, a number of so-called “candidate documents” are extracted. As the number of these reference documents typically is very high (i.e., all documents in the WWW), the efficiency of this step is of high importance. The challenge is to extract a possibly small number of documents for further investigation, without neglecting potentially relevant documents (high recall). In a second step, the so-called *text alignment* procedure, the preselected candidate documents are examined for text fragments that also appear in the suspicious document. In this step, a mapping between text fragments in the suspicious and the candidate set is performed. After this mapping process, a final knowledge-based process is performed, by means of which overlapping text fragments are combined or deleted and visualized.

In practice, it is required to distinguish between two possible scenarios. In the first case, the algorithm has access to a complete data pool. For example, this is the case in a well-defined research area where all relevant literature is stored in a local database. In this case, the algorithm is in full control of the complete procedure of plagiarism search. This scenario is not very likely. In the second more likely case, the first step depends on the utilization of an internet search engine, such as Google or Yahoo!.

The structure of the paper is as follows: In the next section we give a brief overview of the state of the art in plagiarism detection. Then in Section III we give an introduction of Zipf’s Law and the size of typical vocabularies. After that, we present our approach (section IV) and in Section V we provide an algorithm for this approach. We conclude our paper with a number of tasks we plan for the future.

### II. STATE OF THE ART

Since 2009, a competition has been organized in the context of the “Conference and Labs of the Evaluation Forum” (CLEF). The Plagiarism Analysis, Authorship Identification, and Near Duplicate Detection) (PAN) competition uses a standardized collection to compare the different approaches. In 2013, 32 teams joined the competition, in which a number of plagiarized documents was to be detected by the developed software systems. In the following paragraphs, we focus on some of the different approaches according to the different tasks of a Plagiarism Detection System (PDS):

#### A. Source Retrieval

In source retrieval the following steps can be distinguished [2]: In order to generate the search request, the document under evaluation will be chopped into a number of paragraphs (chunking). For example, this fragmentation can be based on chapters or a defined number of lines or sentences. Often, dynamic chunking techniques are utilized. For example, intrinsic methods / procedures which search the document for conspicuities, such as lexis or average word length, in order

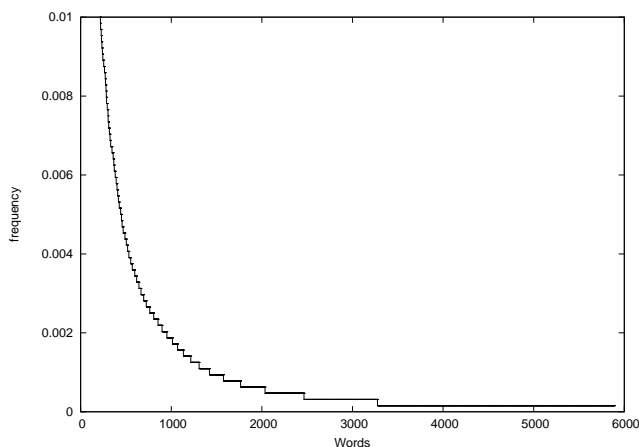


Figure 1. Zipf distribution of the words in all papers of the DBKDA 2013 conference

to find variations leading to potential plagiarized fragments in the document. After completing the first step, the chunks will be used to create keywords or key phrases which then will be implemented in the request for the search engine. This step is typically carried out by i.e. observation of the  $td*idf$  values [3] for single words within the chunks or by the generation of one or more so-called n-grams (phrases with a typical length of 5 to 8 words). These keywords or key phrases will then be submitted to a search engine. Based on the response from the search engine, a number of documents will be downloaded and analyzed in the following step.

### B. Text Alignment

The main purpose of the text alignment step is to identify text fragments from the source data / documents that found their way into the document under investigation. Not only 1:1 plagiarized text fragments should be identified, the objective is to identify disguised or faked fragments as well. These faked phrases are often characterized by simply changing the sequence of words, adding or deleting single words or combining sentences.

### C. Post Processing and Final Visualization

After this mapping process, a final knowledge-based process is carried out in which overlapping text fragments are combined or deleted and accordingly visualized.

## III. WORDS, SENTENCES, AND LANGUAGE

### A. Zipf's Law

Zipf's law [4] postulates that the frequency of any word in a language is inversely proportional to its rank in the corresponding frequency table. Figure 1 shows the distribution of the words that appeared in the papers of the DBKDA-2013 conference [5]. It can be seen that, the distribution is represented by a hyperbolic function.

Examining the articles of the DBKDA 2013 conference, we made the following findings:

- Altogether, about 7000 different (stemmed) words are used.

- The average size of an article is about 5000 words.
- A typical article only contains about 1000 different (stemmed) words.
- The average length of a sentence was between 14 and 22 words for the different articles.
- A paragraph comprised between 7 and 14 sentences for the different articles on the average.

These values were not specific of the conference selected, but can be reproduced with other scientific publications as well. The average length of a sentence in one of the famous literature books *Moby Dick* or *The Adventures of Tom Sawyer* was only about 7 words.

Based on the finding of Zipf's law, it can be postulated that there are words that are more important to identify a document than other words. For example, when we used the following six keywords "the, of, and, in, to, for" in a google query, we got about 11.5 billion results. When searching for the three words "bitmap, index, encoding", only 1.4 million results were obtained. Because a lot of words are inflections or derived from a base form, a stemming process in general reduces all the words to their (morphological) stem. This process reduces the number of different words, without losing the meaning when used in an information retrieval process. The porter stemmer [6] is the most well-known stemmer for the English language.

### B. Size of Vocabulary

The vocabulary of a native English speaker is about 10,000 to 12,000 words. This is quite small compared to the size of the entire English vocabulary which is between 500,000 to 600,000 words (twice the size of the French vocabulary which has only about 300,000 words). This does not include technical terminology. The chemical nomenclature terminology, for example, contains at least 20 million words [7].

## IV. CHOSEN APPROACH

### A. Representation of Text

In this subsection, we examine how the basic units of a document (word, n-gram, sentence, paragraph, whole document) can be represented to support the work of a PDS. The task of a PDS is to find overlapping sections between the suspicious document and the candidate documents. In the trivial case of a 1:1 copy & paste plagiarism, n-grams (a sequence of  $n$  words) can be used, which slide over the document base to find matching regions (technically, this is typically implemented with an inverted index). This approach works very well if the plagiarized extracts are not too obfuscated by changes in the syntax or exchange of words (synonyms, hypernyms).

Another approach is to use the well-known vector space model (VSM). In this model, the whole document or a part of it is represented as an  $n$ -dimensional vector. The vector space is defined by the used vocabulary (words) in the document base. The similarity between two documents (or parts of it) is defined by a similarity function based on the vector representation. A typical similarity function, for example, is the cosine similarity between two vectors. Often, VSM is combined with the  $td*idf$  weighting.



A major difference between the  $n$ -grams representation and the VSM approach is that in VSM the words are represented as an unordered collection. This makes this approach more robust against syntactic modifications of the text. A serious drawback of this approach is that when using a single vector for one document, small plagiarized fragments can easily be overlooked [8]. This can be compensated by building vectors from smaller fragments of the document, which, on the other hand, makes the computation more expensive, because more vectors have to be compared.

### B. Compressed Bitmaps

In our approach, text fragments are represented as compressed bitmaps. Every bit in the bitmap represents a fixed word of our language (i.e., all words sorted alphabetically). Setting the bit at position  $i$  to 1 means, that the  $i^{\text{th}}$  word appears in the given text fragment. As in the case of the VSM, we maintain an unordered list of words (“Bag of Words”) for each text fragment. But, in contrast to VSM, we do not keep the information of how often a word appears in text fragment, we only keep the information whether it appears or not. This seems to be a substantial loss of information at first, but for small units of text (like sentences) the difference is not so big, because in general, most of the words (and especially the relevant words) would not appear more than once anyway. The advantage of our approach is the very fast computation of similarity between two bitmaps using the Jaccard similarity coefficient [3] (equation below).

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}.$$

In our case, the sets  $A$  and  $B$  represent the words in two text fragments. The Jaccard measure is then defined as the coefficient between the number of elements in the intersection of the two sets and the number of elements in the union of the sets. If both sets are equal (i.e., they contain the same words), the value is 1, in the case of no common words the coefficient is 0. In our implementation using bitmaps, the operation consists of an `and` and an `or` operation between two bitmaps and an integer division.

When the chunks of text are small (i.e., sentence or paragraph size), the amount of 1-bits is quite small and the bitmap can be compressed very effectively using Run Length Encoding (RLE) [9]. The required operation, namely, the computation of the Jaccard similarity coefficient can also be performed on the compressed bitmaps, even with higher speed. One important question is how many words the vocabulary should contain? Our experiments with the DBKDA conference proceedings suggest, that probably 10,000 words should be enough. But on the other hand, the whole English vocabulary contains about 500,000 words and the technological terminology can be even much bigger. So in a next experiment, we compare the memory consumption between uncompressed and compressed bitmaps and vector representations. In this experiment, we use different vocabulary sizes starting from 10,000 words up to 5,000,000 words as well as different sizes of text fragments, ranging from 15 words (a typical sentence) to 200 words (a paragraph) up to 5000 words (average paper size). The size of the vector (sparse vector implementation) was calculated by the number of used dimensions in the vector, multiplied by the size of two integers ( $2 * 16$  bit). Figure 2 shows the results of this experiment.

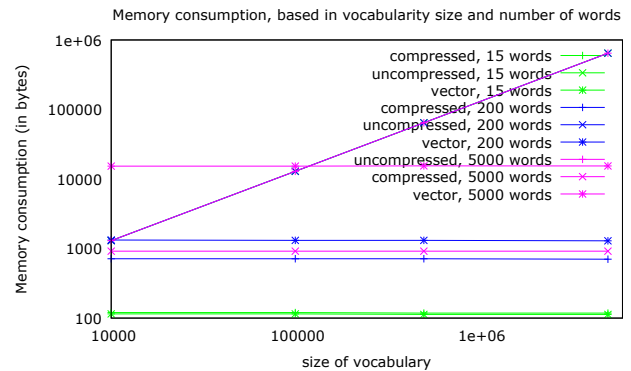


Figure 2. Memory consumption for different representation forms

The most relevant findings of this experiment are:

- The memory consumption for the uncompressed bitmap grows linearly with the size of the vocabulary.
- The memory consumption for the compressed bitmaps and the sparse vector are independent of the vocabulary.
- The size of the compressed bitmaps and the vector depends on the length of the text fragment.
- For the sentence case (15 words), vector and compressed bitmap have nearly the same memory consumption (vector 6% less). For the 200-word paragraph, the bitmap only needs 53% of the memory of the vector representation. For the full paper (5000 words) case, the compressed bitmap memory consumption is only about 6% of the vector representation.

### C. Influence of Word Ordering in Bitmap

The position of the words in the bitmap has an influence on the size of the compressed bitmaps. Figure 3 shows two different ordering schemes. In the first (sentence with about 15 words) and third bitmaps (paragraph with 100 words), the words are ordered alphabetically, which results in a nearly equal distribution (the vertical lines represent 1-bits). In the second and fourth bitmaps, by contrast, the order is based on the Zipf distribution shown in Figure 1. Words with a very high frequency appear at the beginning of the bitmap and the words with low frequency appear at the end of the bitmap.

The ordering based on the Zipf distribution has two advantages:

- 1) The gaps between two 1-bits are very small at the beginning of the bitmap, but grow towards the right. Based on the characteristics of the Word Aligned Hybrid (WAH) algorithm, which needs at least a gap size of a multiple of 31 (or 63), this leads to a number of fill words at the beginning of the bitmap, followed by 0-fills with growing capacity, interrupted by single literals. This results in a better compression ratio as when the set bits are more evenly distributed over the whole bitmap. Figure 4 compares the memory consumption for the different sorting orders. On the

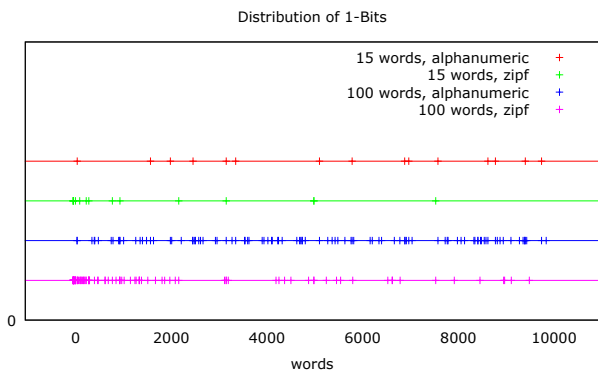


Figure 3. Comparison of distributions using alphabetic or word frequency-based ordering of words in bitmap

sentence (15 words) and paragraph (200 words) level, the amount of memory for the Zipf-based sorting is about 75% compared to alphabetic ordering. The gain of memory for the 5000 words case is smaller (about 2%).

- As discussed before, words with high frequency are not very usable to identify documents. As these words are now grouped together at the beginning of the bitmap, they can simply be ignored by skipping the first  $n$  bits.

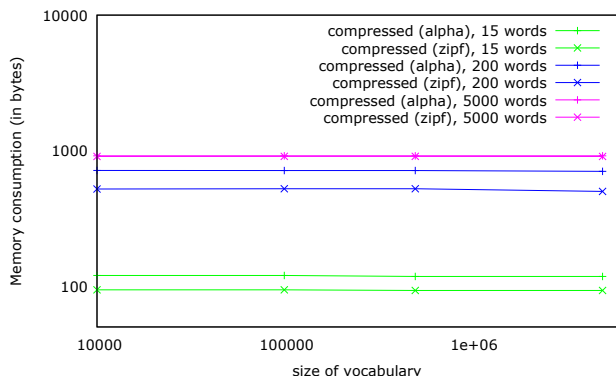


Figure 4. Comparison of distribution using alphabetic or word frequency based ordering of words in bitmap

#### D. Comparison of Execution Time for Measuring Similarity

In a last experiment, we compare the time for the computation of the similarity measure using different measure functions and text representations. Figure 5 shows the difference in execution time between the cosine measure based on the vector representation and the Jaccard measure based on compressed and uncompressed bitmaps. The main findings of this experiment are:

- The computation of the similarity measure using compressed bitmaps and the Jaccard coefficient is much more independent of the chunk size (sentence, paragraph, whole paper) compared to the cosine similarity measure. The time difference using the bitmaps is in a range of a factor of 2 compared to three orders

of magnitude for using the vectors with the cosine similarity measure.

- The performance of the cosine similarity is slightly better (max. factor of 2) for the sentence case. In the case of considering a whole paragraph or paper, the bitmaps solution is at least superior by a factor of 4, growing up to a factor of 50.

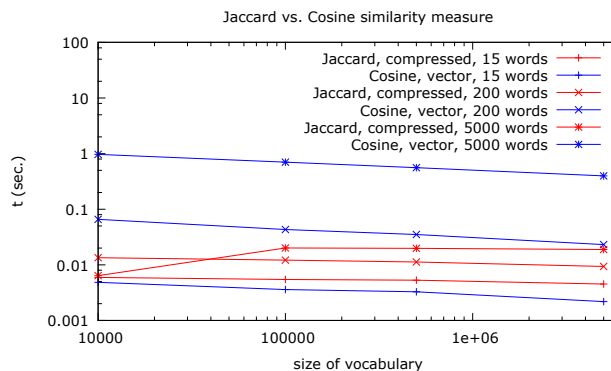


Figure 5. Comparison of execution time for similarity measure based on compressed bitmaps (Jaccard) and vector (cosine measure)

Based on the findings of the last two experiments, the usage of bitmaps seems an appropriate choice for the *source retrieval* step, to find the “candidate set”. In contrast to the much more expensive cosine measure which typically only allows to use one vector per document, it is possible to subdivide a document into a number of smaller parts which can be examined separately and, hence, to minimize the chance to overlook smaller parts, which have been plagiarized.

But also for the *text alignment* step, compressed bitmaps seem to be suitable. The reason for this is, that compared to the search for  $n$ -grams which require that the order of the words is the same in the suspicious document as well as in the candidate documents, the order of the word is irrelevant. This makes this approach insensitive to obfuscation approaches like paraphrasing single sentences. Additionally, the obfuscation approach by replacing single words by synonyms or hypernyms can be handled easily. In this case, not only the bit for a concrete word has to be set, but also for possible synonyms and hypernyms. These words can be provided automatically using Wordnet [10].

#### V. ALGORITHM

The algorithm for the identification of the candidate set is as follows: In a preprocessing step, all documents which form the comparison document set are fragmented into a small number of chunks. For each of these text-fragments the bitmap representation is built. Additionally, the amount of 1-bits (the number of words) is stored. To handle obfuscations, taxonomies from Wordnet are used and for every word where Wordnet offers a synonym or one or more hypernyms, the bits for these words are also set (the number of words determined previously is not incremented). After this enrichment step, the bitmaps are compressed using the WAH algorithm [11]). Parallel to the fragmentation of the document into a small number of text fragments for the candidate search, a sentence-wise

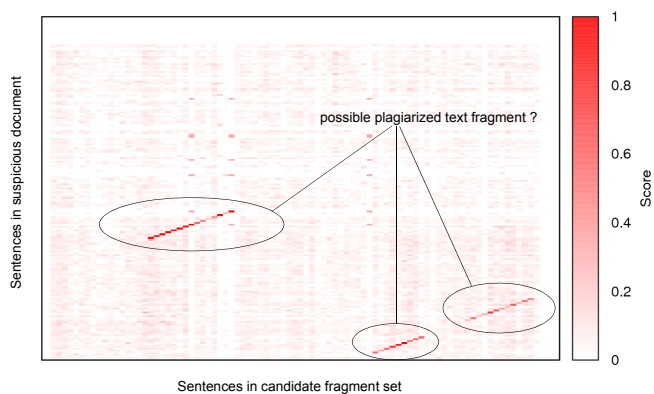


Figure 6. Plagiat Matrix

fragmentation and transformation into compressed bitmaps is performed for the later text alignment step. In the same way, but without the enrichment of the Wordnet taxonomies, the suspicious document is preprocessed.

In the next step, every compressed bitmap of the suspicious document is compared with every bitmap of the document base using the Jaccard measure coefficient. As we are only interested in finding the candidate set, the most leftmost bits (which represent the most irrelevant words) can be ignored. If the Jaccard measure is above a threshold value, the fragment the bitmap belongs to is included into the candidate set.

In the text alignment step, the bitmaps representing the sentences are considered. Every compressed bitmap representing a sentence in the suspicious document is compared using the Jaccard measure with all sentences from the fragments identified in the first step. As a result, we get a matrix, where each row represents a sentence in the suspicious document and each column represents a sentence from the qualified fragments. Every cell in the matrix has a value between zero and one, representing the similarity between two sentences. This matrix can easily be represented graphically using a heatmap, as it is shown in Figure 6. Using a color gradient for the values in the interval  $[0, 1]$  from white to red, we can easily identify fragments with similar or alike content. The lines originate from a number of consecutive sentences with high similarity and, therefore, are probably plagiarisms. The representation of plagiarized fragments as lines is also shown in [12].

## VI. CONCLUSION

We presented a new approach to plagiarism detection using compressed bitmaps. As we have shown, the bitmap approach can be used for the candidate retrieval as well as for the text alignment process. At the beginning of the paper, we show that from the memory consumption aspect and the performance aspect, a compressed bitmap with the Jaccard measure is superior to the vector representation using cosine similarity measure. To cover both steps, we build compressed bitmaps based on different aggregation levels. An additional enrichment step using semantic taxonomies from Wordnet allows us to also cover obfuscation techniques like renaming words. Obfuscation by paraphrasing is also covered by our approach, based on the set characteristic (no order of words). The final

visual representation using heatmaps shows plagiarized text fragments as lines.

As a next step, we have to finely tune our process, find appropriate threshold values, and compare our results with others (see PAN competition mentioned in Section II). Another interesting aspect for our future research is the parallelization of the whole process using a framework like Hadoop [13].

## REFERENCES

- [1] B. Stein, S. M. zu Eissen, and M. Potthast, "Strategies for retrieving plagiarized documents," in Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ser. SIGIR '07. New York, NY, USA: ACM, 2007, pp. 825–826.
- [2] M. Potthast, M. Hagen, T. Gollub, M. Tippmann, J. Kiesel, P. Rosso, E. Stamatatos, and B. Stein, "Overview of the 5th international competition on plagiarism detection," in CLEF 2013 Evaluation Labs and Workshop Working Notes Papers, 2013.
- [3] C. D. Manning, P. Raghavan, and H. Schütze, Introduction to Information Retrieval. New York, NY, USA: Cambridge University Press, 2008.
- [4] G. Zipf, Human behavior and the principle of least effort: an introduction to human ecology. Addison-Wesley Press, 1949. [Online]. Available: <http://books.google.de/books?id=1tx9AAAAIAAJ>
- [5] IARIA, "ThinkMind // DBKDA 2013, The Fifth International Conference on Advances in Databases, Knowledge, and Data Applications," 2013, [accessed 24-Feb-2014]. [Online]. Available: <http://www.thinkmind.org/index.php?view=instance&instance=DBKDA+2013>
- [6] M. F. Porter, "Readings in information retrieval," K. Sparck Jones and P. Willett, Eds. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, ch. An Algorithm for Suffix Stripping, pp. 313–316.
- [7] Wikipedia, "Wortschatz — Wikipedia, the free encyclopedia," 2013, [accessed 24-Feb-2014]. [Online]. Available: <http://de.wikipedia.org/wiki/Wortschatz>
- [8] N. Meuschke and B. Gipp, "State of the Art in Detecting Academic Plagiarism," International Journal for Educational Integrity, vol. 9, no. 1, Jun. 2013, pp. 50–71.
- [9] M. Nelson, The Data Compression Book. New York, NY, USA: Henry Holt and Co., Inc., 1991.
- [10] G. A. Miller, "Wordnet: A lexical database for english," Communications of the ACM, vol. 38, 1995, pp. 39–41.
- [11] K. Wu, E. J. Otoo, and A. Shoshani, "Optimizing bitmap indices with efficient compression," ACM Trans. Database Syst., vol. 31, no. 1, 2006, pp. 1–38.
- [12] T. Gottron, "External plagiarism detection based on standard ir technology and fast recognition of common subsequences - lab report for pan at clef 2010," in CLEF (Notebook Papers/LABs/Workshops), M. Braschler, D. Harman, and E. Pianta, Eds., 2010.
- [13] T. White, Hadoop: The Definitive Guide, 1st ed. O'Reilly Media, Inc., 2009.

# Enriching Dimension Hierarchies with Topological Relations to Improve the Development of Spatial Data Warehouse

Sana Ezzedine

LTSIRS Laboratory, National School of Engineers,  
University of Tunis El Manar, Tunisia

[sana.ezzedine@gmail.com](mailto:sana.ezzedine@gmail.com)

Sami Yassine Turki and Sami Faiz

LTSIRS Laboratory, National School of Engineers,  
University of Tunis El Manar, Tunisia

[yassine.turki@isteub.rnu.tn](mailto:yassine.turki@isteub.rnu.tn) and  
[sami.faiz@insat.rnu.tn](mailto:sami.faiz@insat.rnu.tn)

**Abstract**—The design of a spatial data warehouse depends on operational data, spatial and non spatial requirements in order to support the decision-making process required by final users. It is crucial to consider decision maker requirements in the conceptual level of the construction of a spatial data warehouse. Furthermore, updating a spatial data warehouse and especially the addition of a new user's requirement after the construction of a spatial data warehouse is a great need for users. In this paper, to overcome this problem, dimension hierarchies will be specified in the Spatial Data Warehouse using topological relationships among spatial objects. Dimension hierarchies added show spatial requirements which are necessary to improve decision-making process. Decision makers thus will be able to achieve their information needs for analysis. Finally, we show the benefits of our approach by providing a case study, which defines an enriched conceptual model of a spatial data warehouse.

**Keywords**—spatial data warehouse; updating; spatial requirements; decision-making.

## I. INTRODUCTION

If we consider the definition proposed by Inmon [1], a Spatial Data Warehouse (SDW) is “a subject oriented, integrated, non-volatile, and time variant collection of spatial data in support of management's decision”.

The design of a SDW is based on a Multidimensional Model, which contains facts and dimensions. Facts contain the business metrics (i.e., measures) and dimensions describe facts and context to analyze these facts using dimension attributes organized in hierarchies.

Several approaches are proposed to model the design of a SDW. They did not define formal and standard transformations between the design and the implementation of SDWs in a specific platform. Moreover, they did not suggest an automatic transformation from the conceptual model to the possible logical representation. In addition, they did not consider needs related to the spatial Decision Maker (DM)'s requirements. To overcome these problems, we defined an approach [2] based on the Model Driven Architecture (MDA) models and the Unified Modeling Language (UML). This approach considers both spatial and non spatial requirements, described by a Geographic

Computation Independent Model (Geo CIM), the first MDA Model. The Geo CIM is integrated by means of transformation rules into a Geographic Platform Independent Model (Geo PIM), the second MDA model, which defines the conceptual Multidimensional model of a SDW.

Within this approach, once user requirements are correctly captured, we obtain automatically the corresponding Multidimensional and conceptual model of a SDW. Nevertheless, in this approach, we find that the required multidimensional model does not take account of the updating requirements of a DM. Therefore, the final SDW will not completely satisfy final user requirements.

Our aim is to improve the quality of dimension hierarchies by means of adding new hierarchy aggregation levels, which allow SDW DMs to achieve their analysis information needs [2]. Dimension hierarchies enable also to the adding of new requirements to better support the decision-making process.

In this paper, we present an approach that treats updating in terms of adding new spatial requirements. We propose to enrich dimension hierarchies by adding new levels of aggregation in order to obtain the required hierarchies.

To accomplish this, we propose the use of semantic relations among spatial concepts provided by topological relationships [3]. The initial hypothesis is that both SDWs and Topological relationships present hierarchical structures: dimension hierarchies in SDWs show the relationships between value domains from different dimension attribute (set by levels of aggregation) [4], while topological relations present hierarchical semantic relations between spatial concepts, such as adjacency or inclusion or intersection, etc. [3]. Therefore, our approach is based on using these topological relations to add new levels to dimension hierarchies in order to obtain the required hierarchies. Figure. 1 summarizes this scenario.

The remainder of this paper is structured as follows. Section 2 presents an overview of works about the development of SDWs and the addition of dimension Hierarchies in the conceptual level. Section 3 defines our approach for enriching dimension hierarchies using topological relationships. In Section 4, a case study is presented. Finally, we point out our conclusions and sketch some future work in Section 5.

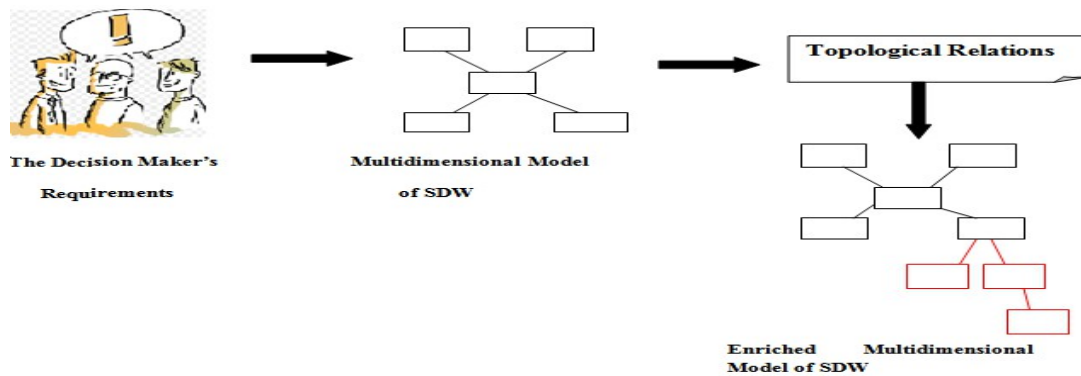


Figure 1: Using Topological Relations to enrich the Multidimensional Model of SDW

## II. RELATED WORK

It is widely accepted that the development of SDWs must be based on a conceptual model. Therefore, in this section, we focus on briefly describing the most relevant approaches for the conceptual modeling of SDWs and, more generally, the addition of dimension Hierarchies in conceptual modeling.

### A. Conceptual Modeling of SDWs

Various approaches for the conceptual design of SDW systems have been proposed in the last few years. In this section, we present a brief discussion about some of the most well-known approaches.

The first attempt integrated spatial information and ensured correct aggregation over spatial [4][5]. Other works defined a multidimensional analysis tool that modeled spatial Data in a SDW [6][7][8][9]. Alternatively, authors defined a query language [10][11] that allowed the use of multidimensional and spatial and topological operators such as GeoMDQL [12]. All these approaches did not present an unequivocal and automatic transformation to every possible logical representation from a conceptual model. In addition, they did not consider needs related to the DM's requirements.

More recently, some approaches have tried to overcome these limitations, especially the problem of the automatic transformations. These approaches used standards framework as MDA. MDA provides a set of guidelines to structure specifications expressed as models. An alignment of multidimensional spatial model with MDA is proposed in [13]. The same approach is extended [14] to include spatial data in the SDW design level. It allows DM to define his geographical queries independently of the logical presentation. [15] proposed to consider the DM's aims and defined [16] some spatial elements describing the top DM's goals. In the same context, a Case tool based on Unified Modeling Language (UML) standard is used by [17] to model both spatial and non spatial data in the SDW design. [18] focused on the use of transformations based on MDA to automatically generate the data and the analysis models.

Every of the above-described approaches presented conceptual models lacked the integration of all DMs' spatial needs in the SDW design. To overcome this problem, [2] proposed an approach, which aims to integrate DM's requirements in the SDW Design. It

presented an approach that automatically generates the design of SDW from a requirement's model.

This approach does not consider updating requirements in terms of adding new contexts to the requirements model after the development of SDW. This must be taken into account in stages of the development process, i.e., the conceptual modeling of the SDWs.

### B. Adding Dimension Hierarchies in SDWs

Mazón and Trujillo [19] suggested enriching dimension hierarchies in terms of structure and data. They considered dimension hierarchy as semantic relationships between values and they proposed to exploit the hypernymy/hyponymy relationships ("is-a-kind-of") and Meronymy/Holonymy ("is-a-part-of") WordNet. In this approach, levels of granularity are created at the end of hierarchy.




Favre et al. [20] proposed to enhance the dimension hierarchies by exploiting the knowledge of users. This knowledge is represented by a meta-aggregation rule and different rules. A meta-aggregation rule represents the structure of the link aggregation between two levels of granularity. And rules "if-then" represent the link at the instances. The levels created can be inserted into a hierarchy or created at the end thereof.

## III. USING TOPOLOGICAL RELATIONSHIPS TO ENRICH DIMENSION HIERARCHIES

Dimension hierarchies in SDWs show the relationships between domains of values from different dimension attributes (set in levels of aggregation). Topological relationships also present hierarchical relationships between spatial concepts, such as adjacency and connectivity. Thereby, we use topological relationships to automatically complete dimension hierarchies in a conceptual model of a SDW.

In this paper, we use classes' stereotypes defined in [2]. These classes are based on Unified Modeling Language (UML) as shown in Table 1.

TABLE 1. STEREOTYPES USED TO DEVELOP THE CONCEPTUAL MODEL OF A SDW

Stereotype	Description	Presentation
Fact Class	Facts contain business measures	
Dimension Class	Dimensions describe Facts	
Base Class	Base represent Dimension Hierarchy with their attributes	

In this paper, we define another stereotype based also on UML named Spatial Hierarchy, as shown in Table 2.

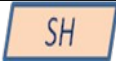
Spatial Hierarchy is added in the conceptual model of a SDW when the DM needs to take account of a new context of spatial requirements in the developing of the SDW.

Our proposal consists of identifying topological relationships between existing dimensions and bases in the conceptual model of SDW and the new added spatial requirements given by the user.

With each identified topological relationship, we create a Spatial Hierarchy, which is named with the same name of the identified topological relationship and has as attributes the characteristics of the added requirement.

Following, we explain the main steps of our approach (an overview is shown in Figure. 2).

TABLE 2. SPATIAL HIERARCHY STEREOTYPE USED TO DEVELOP THE CONCEPTUAL MODEL OF A SDW

Stereotype	Description	Presentation
Spatial Hierarchy	Spatial Hierarchy present spatial Dimension hierarchy with their attributes.	

Prerequisite 1. A dimension attribute is chosen from the initial conceptual model of the SDW. The spatial hierarchy will be enriched starting from this attribute.

Prerequisite 2. A new spatial requirement has been added by the DM, which is in relation with one of existing dimensions in the initial conceptual model.

Step 1. Extract different instances from the dimension attribute chosen from the initial conceptual model.

Step 2. Identify topological relationships between spatial objects recently required with spatial objects existing in the dimension attribute chosen.

Step 3. If there are relationships between the required spatial objects and the existed ones, a spatial hierarchy for every relationship is created having the same name as the topological relationship.

Step 3'. If there are no relationships between the required spatial objects and the existed ones, a new record is inserted in the selected dimension without creating a new hierarchy.

In Figure. 2, every step of our approach is illustrated. From a dimension or a dimension hierarchy in a Multidimensional model, which not accomplishes all user requirements, a dimension attribute is chosen. Then the topological relationships are identified between instances of the dimension attribute chosen and the new requirement in order to create a new level of the spatial dimension hierarchy. If there are no relationships between added requirement and existed dimensions, a new record is inserted. Iterations are repeated until all required spatial objects are classified.

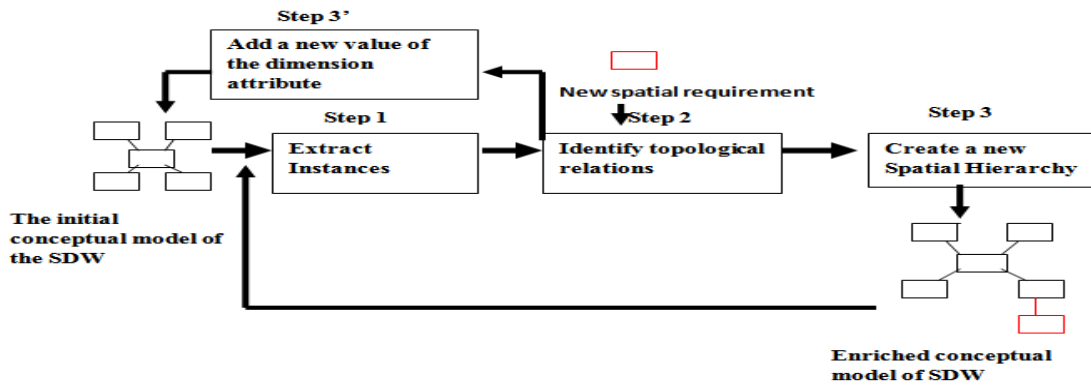


Figure 2. Overview of our approach

#### IV. CASE STUDY

In this section, we show the benefits of our approach by providing a case study, in which Spatial hierarchies are enriched in the conceptual Model of a SDW. Our case study consists of defining a conceptual model of a sales manager who wants to analyze sales operations in stores situated 2 km around the airport. Then, he needs to extend the analysis region by adding others streets.

The initial conceptual model before adding spatial requirements and applying our approach is presented in Figure. 3. In this case, the added requirements are the extended streets.

As is described previously, we should choose firstly a dimension attribute. In this case, we choose the dimension spatial cover and the dimension attribute spatial objects. Then we identify relations between spatial objects and streets added. We found that some streets have an intersection relationship with existed streets; others have an inclusion relationship and others streets having no relationships with existed spatial objects.

The conceptual model is extended according to the different stages of the approach as shown in Figure. 4.

The initial conceptual model contains facts and dimensions presenting the spatial and non spatial requirements. The facts presented in Figure. 3 are: Sale and Spatial Cover. The dimensions are: Product, Time, Operational, DMcharacteristic, Presentation and Semantic.

Figure.3 presents the spatial and non-spatial data that are necessary for the decision maker, the sales manager, to make the right decision. We use classes of the Unified Modeling Language UML to model

requirements. Each class is a one of the requirements expressed by the sales manager.

After the addition of new spatial requirements, we identify two types of relations, as shown in Figure.4, inclusions' relations and intersections' relations between spatial objects and added spatial objects.

Consequently, two spatial hierarchies are added according to the two relationships identified.

#### V. CONCLUSION AND FUTURE WORK

Spatial Dimension hierarchies are important to support the decision making process, since they allow the analysis of data at different levels of detail (i.e., levels of aggregation). Then, obtaining the required spatial hierarchies captured from decision maker is crucial for specifying a successful SDW.

In this paper, we propose the application of topological relationships to obtain the required hierarchies. The advantage of our proposal is clear: the enrichment of the conceptual model of the SDW by adding new aggregation levels in order to satisfy the required DM requirements.

These required hierarchies allow SDW users to satisfy their information analysis needs, since they better support the decision-making process.

Our proposal can be generalized to generate a star scheme or a snowflake scheme of a conceptual model of a SDW appropriate for a group of DMs.

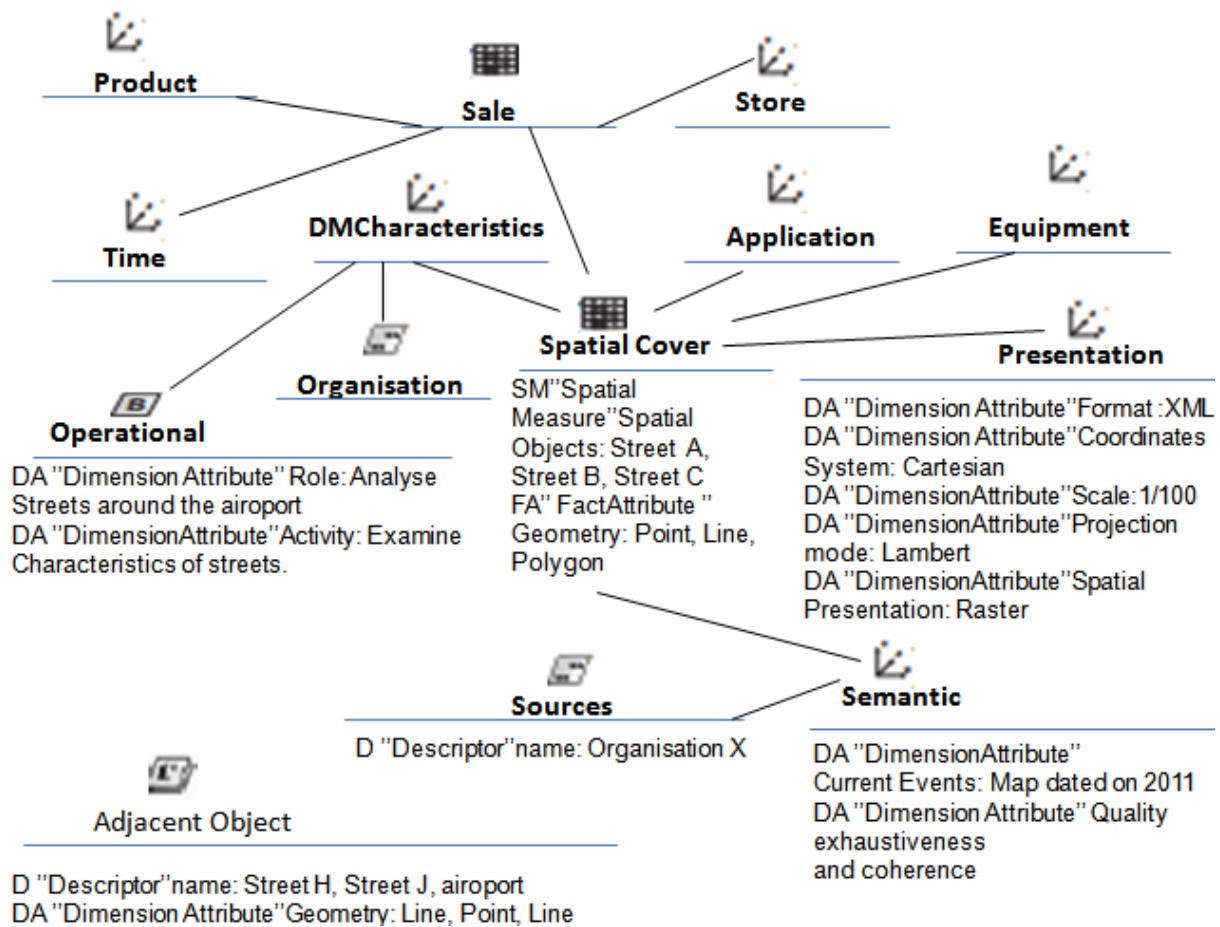


Figure 3. The conceptual model of the SDW used by the sales manager [2]

REFERENCES

[1] W. Inmon, "Building the Data warehouse". In John Wiley & Sons, Chichester , 1996.

[2] S. Ezzedine, S. Y. Turki, and S. Faiz "An approach based on Model Driven Engineering for data integration in a spatial data warehouse", in CODIT'13, 2013.

[3] J. Egenhofer and R. D.Franzosa "Point-Set Topological Spatial Relations", in International Journal for Geographical Information Systems, vol. 5, pp. 161-174, 1991.

[4] S. Bimonte, A. Tchounikine, and M. Miquel, "Geocube, a multidimensional model and navigation operators handling complex measures: Application in spatial olap", Advances in Information Systems (ADVIS), pp. 100-109, Berlin / Heidelberg, Germany, 2006.

[5] S. Bimonte, A. Tchounikine, and M. Bertolotto "Integration of geographic information into multidimensional models", in ICCSA, pp. 316-329, 2008.

[6] A. Escribano, L. Gomez., B. Kuijpers, and A. Vaisman "Piet: a gis-olap implementation", in DOLAP '07: Proceedings of the ACM tenth international workshop on Data warehousing and OLAP, pp 73-80, New York, NY, USA, 2007.

[7] L. Gomez, S. Haesevoets, B. Kuijpers, and A. Vaisman "Spatial aggregation: Data model and implementation", in CoRR, abs/0707.4304, 2007.

[8] E. Malinowski and E. Zimanyi "Representing spatiality in a conceptual multidimensional model", in GIS '04: Proceedings of the 12th annual ACM international workshop on Geographic information systems, pp. 12-22, New York, NY, USA, 2004.

[9] E. Malinowski and E. Zimanyi "Implementing spatial datawarehouse hierarchies in object-relational dbms", in ICEIS, pp. 186-191, 2007.

[10] J. Da Silva, V. C. Times, A. C. Salgado, C. Souza, R. do Nascimento Fidalgo, and A.G. de Oliveira "Querying geographical data warehouses with geomdql", in SBBD, pp. 223-237, 2007.

[11] N. Stefanovic, J. Han, and K. Koperski "Object-based selective materialization for efficient implementation of spatial data cubes", in IEEE Transactions on Knowledge and Data Engineering, vol. 12. N. 6, pp. 938-958, 2000.

[12] S. Rivest, Y. Bedard, and P. Marchand "Toward better support for spatial decision making: Defining the



characteristics of spatial on-line analytical processing”, in Geomatica, vol. 55, N. 4, pp.539–555, 2001.

[13] O. Glorio and J. Trujillo, “An MDA Approach for the Development of Spatial Data Warehouses”, In DaWaK '08, 2008.

[14] O. Glorio and J. Trujillo, “Designing Dhata Warehouses for Geographic OLAP querying by using MDA”, in ICCSA '09, 2009.

[15] J. Mazon and J. Trujillo, “A Hybrid Model Driven Development Framework for the Multidimensional Modeling of Data Warehouses”, in SIGMOD, 2007.

[16] O. Glorio, J. Mazón, I. Garrigós, and J. Trujillo, “Using Web-based Personalization on Spatial Data Warehouses”, in EDBT '10, 2010.

[17] R. Fidalgo and A. Cuzzocrea, “An Enhanced Spatial Data Warehouse Metamodel”, in CAiSE Forum, Vol. 855 of CEUR Workshop Proceedings, pp.32-39, 2012.

[18] A. Cuzzocrea, J. Mazon, J. Trujillo, and J. Zubcoff “Model-driven data mining engineering: from solution-driven implementations to 'composable' conceptual data mining models”, in Int. J. of Data Mining, Modelling and Management, 2011 Vol.3, No.3, pp.217 – 251, 2011.

[19] J.N. Mazón, and J. Trujillo, “Enriching Data Warehouse Dimension Hierarchies by Using Semantic Relations”, in XXIIIrd British National Conference on Databases (BNCOD 2006), Vol. 4042, pp. 278–281. Springer, 2006.

[20] C. Favre, F. Bentayeb, and O. Boussaïd, “Dimension Hierarchies Updates in Data Ware-houses : a User-driven Approach”, in IXth International Conference on Enterprise Informa-tion Systems (ICEIS 07), Funchal, Madeira, Portugal, 2007.

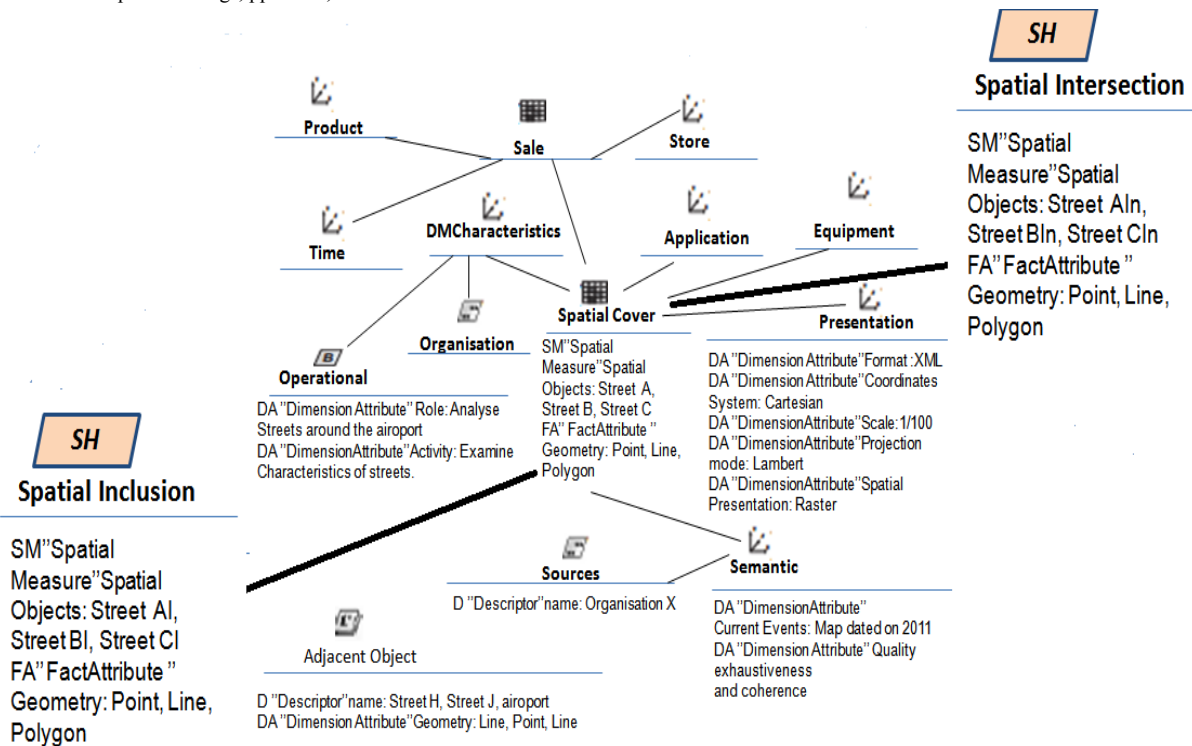


Figure 4. The extended conceptual model of the SDW after adding spatial requirements

# In-Memory Distance Threshold Queries on Moving Object Trajectories

Michael Gowanlock

Department of Information and Computer Sciences and  
NASA Astrobiology Institute  
University of Hawai'i, Honolulu, HI, U.S.A.  
Email: gowanloc@hawaii.edu

Henri Casanova

Department of Information and Computer Sciences  
University of Hawai'i, Honolulu, HI, U.S.A.  
Email: henric@hawaii.edu

**Abstract**—The need to query spatiotemporal databases that store trajectories of moving objects arises in a broad range of application domains. In this work, we focus on in-memory distance threshold queries which return all moving objects that are found within a given distance  $d$  of a fixed or moving object over a time interval. We propose algorithms to solve such queries efficiently, using an R-tree index to store trajectory data and two methods for filtering out trajectory segments so as to reduce segment processing time. We evaluate our algorithms on both real-world and synthetic in-memory trajectory datasets. Choosing an efficient trajectory splitting strategy to reduce index resolution increases the efficiency of distance threshold queries. Interestingly, the traditional notion of considering good trajectory splits by minimizing the volume of MBBs so as to reduce index overlap is not well-suited to improve the performance of in-memory distance threshold queries.

**Keywords**—spatiotemporal databases; query optimization.

## I. INTRODUCTION

Moving object databases (MODs) have gained attention as applications in several domains analyze trajectories of moving objects (animals, vehicles, humans, stellar bodies, etc.). Contributing to the motivation for MOD research is the proliferation of mobile devices that provide location information (e.g., GPS tracking). We focus on MODs that store historical trajectories [1], [2], [3], [4], such as the movement patterns of animals over a given period of observation, and that must support queries over subsets, or perhaps the full set, of the trajectory histories. In particular, we focus on two types of *distance threshold queries*:

- 1) Find all trajectories within a distance  $d$  of a given static point over a time interval  $[t_0, t_1]$ .
- 2) Find all trajectories within a distance  $d$  of a given trajectory over a time interval  $[t_0, t_1]$ .

An example query of the first type would be to find all animals within a distance  $d$  of a water source within a day. An example query of the second type would be to find all police vehicles on patrol within a distance  $d$  of a moving stolen vehicle during an afternoon. We investigate efficient distance threshold querying on MODs, making the following contributions:

- We propose algorithms to solve the two types of in-memory distance threshold queries above.
- We make the case for using an R-tree index for storing trajectory line segments.

- Given a set of candidate line segments returned from the R-tree, we propose methods to filter out line segments that are not part of the query result set.
- We propose decreasing index resolution to exploit the trade-off between the amount of index overlap and the number of entries in the index by exploring three trajectory splitting strategies.
- We demonstrate that, for in-memory queries, lower-bounding the index resolution is more important than minimizing the volume of hyperrectangular minimum bounding boxes (MBB), and thus index overlap.
- We evaluate our proposed algorithms using both real-world and synthetic datasets for both 3-D and 4-D trajectory data (i.e., the temporal dimension plus either 2 or 3 spatial dimensions).

This paper is organized as follows. In Section II, we outline related work. Section III defines the distance threshold query. Section IV discusses the indexing method. In Section V we present our algorithms and present an initial performance evaluation in Section VI. Section VII motivates, proposes, and evaluates methods to filter the candidate line segments. Section VIII presents and evaluates methods to split trajectories to reduce index resolution for efficient query processing. Finally, Section IX concludes the paper with a brief summary of findings and perspectives on future work.

## II. RELATED WORK

A trajectory is a set of points traversed by an object over time in Euclidean space. In MODs, trajectories are stored as sets of spatiotemporal line segments. The majority of the literature on indexing spatiotemporal data utilizes R-tree data structures [5]. An R-tree indexes spatial and spatiotemporal data using MBBs. Each trajectory segment is contained in one MBB. Leaf nodes in the R-tree store pointers to MBBs and the segments they contain (coordinates, trajectory id). A non-leaf node stores the dimensions of the MBB that contains all the MBBs stored (at the leaf nodes) in the non-leaf node's subtree. Searches traverse the tree to find all (leaf) MBBs that overlap with a query MBB. Variations of the R-tree and other methods have been proposed (TB-trees [6], STR-trees [6], 3DR-trees [7], SETI [8], and TrajStore [9]).

Few works on trajectory similarity searches have studied distance threshold queries [3]. Other types of trajectories,

which we do not consider in this work, have been studied (e.g., flocks [10], convoys [4], swarms [11]). The well-studied  $k$  Nearest Neighbors ( $k$ NN) queries [12], [13], [14], [15] are related to distance threshold queries. A distance threshold query can be seen as a  $k$ NN query with an unknown value of  $k$ . As a result, previous work on  $k$ NN queries (with known  $k$ ) cannot be applied directly to distance threshold queries (with unknown  $k$ ).

### III. PROBLEM STATEMENT

#### A. Motivating Example

One motivation application for this work is in the area of astrophysics [16]. The past decade of exoplanet searches implies that the Milky Way, and hence the universe, hosts many rocky, low mass planets that may sustain complex life. However, regions of a galaxy, such as the Milky Way, may be inhospitable due to transient radiation events, such as supernovae explosions or close encounters with flyby stars that can gravitationally perturb planetary systems. Studying habitability thus entails solving the following two types of *distance threshold queries* on the trajectories of (possibly billions of) stars orbiting the Milky Way: (i) Find all stars within a distance  $d$  of a supernova explosion, i.e., a non-moving point over a time interval; and (ii) Find the stars, and corresponding time periods, that host a habitable planet and are within a distance  $d$  of all other stellar trajectories.

#### B. Problem Definition

Let  $D$  be a database of  $N$  trajectories, where each trajectory  $T_i$  consists of  $n_i$  4D (3 spatial + 1 temporal) line segments. Each line segment  $L$  in  $D$  is defined by the following attributes:  $x_{start}$ ,  $y_{start}$ ,  $z_{start}$ ,  $t_{start}$ ,  $x_{end}$ ,  $y_{end}$ ,  $z_{end}$ ,  $t_{end}$ , trajectory id, and segment id. These coordinates for each segment define the segment's MBB (note that the temporal dimension is treated in the same manner as the spatial dimensions). Linear interpolation is used to answer queries that lie between  $t_{start}$  and  $t_{end}$  of a given line segment.

We consider historical continuous *searches* for trajectories within a distance  $d$  of a *query*  $Q$ , where  $Q$  is a moving object's trajectory,  $Q_t$ , or a stationary point,  $Q_p$ . More specifically:

- $\text{DistTrajSearch}_{Q_p}(D, Q_p, Q_{start}, Q_{end}, d)$  searches  $D$  to find all trajectories that are within a distance  $d$  of a given query static point  $Q_p$  over the query time period  $[Q_{start}, Q_{end}]$ . The query is continuous, such that the trajectories found may be within the distance threshold  $d$  for a subinterval of the query time  $[Q_{start}, Q_{end}]$ . For example, for a query  $Q_1$  with a query time interval of  $[0,1]$ , the search may return  $T_1$  between  $[0.1,0.3]$  and  $T_2$  between  $[0.2,0.6]$ .
- $\text{DistTrajSearch}_{Q_t}(D, Q_t, Q_{start}, Q_{end}, d)$  is similar but searches for trajectories that are within a distance  $d$  of a query trajectory  $Q_t$ .

$\text{DistTrajSearch}_{Q_p}$  is a simpler case of  $\text{DistTrajSearch}_{Q_t}$ . We focus on developing an efficient approach for  $\text{DistTrajSearch}_{Q_t}$ , which can be reused as is for  $\text{DistTrajSearch}_{Q_p}$ .

In all that follows, we consider *in-memory databases*, meaning that the database fits and is loaded in RAM once and

for all. Distance threshold queries are relevant for scientific applications that are typically executed on high-performance computing platforms such as clusters. It is thus possible to partition the database and distribute it over a (possibly large) number of compute nodes so that the application does not require disk accesses. It is straightforward to parallelize distance threshold searches (replicate the query across all nodes, search the MOD independently at each node, and aggregate the obtained results). We leave the topic of parallel searches for future work. Instead we focus on efficient in-memory processing at a single compute node, which is challenging and yet necessary for achieving efficient parallel executions. Furthermore, as explained in Section IV, no criterion can be used to avoid index tree node accesses in distance threshold searches. Therefore, there are no possible I/O optimizations when (part of) the database resides on disk, which is another reason why we focus on the in-memory scenario.

### IV. TRAJECTORY INDEXING

Given a distance threshold search for some query trajectory over some temporal extent, one considers all relevant query MBBs (those for the query trajectory segments). These query MBBs are *augmented* in all spatial dimensions by the threshold distance  $d$ . One then searches for the set of trajectory segment MBBs that overlap with the query MBBs, since these segments may be in the result set. Efficient indexing of the trajectory segment MBBs can thus lower query response time.

The most common approach is to store trajectory segments as MBBs in an index tree [12], [13], [14], [15]. Several index trees have been proposed (TB-tree [6], STR-tree [6], 3DR-tree [7]). Their main objective is to reduce the number of tree nodes visited during index traversals, using various pruning techniques (e.g., the *MINDIST* and *MINMAXDIST* metrics in [17]). While this is sensible for  $k$ NN queries, instead for distance threshold queries *there is no criterion for reducing the number of tree nodes that must be traversed*. This is because any MBB in the index that overlaps the query MBB may contain a line segment within the distance threshold, and thus must be returned as part of the candidate set.

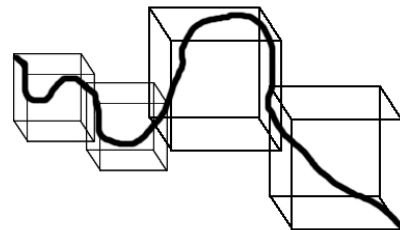


Figure 1. An example trajectory stored in different leaf nodes in a TB-tree.

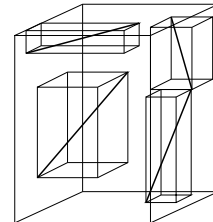


Figure 2. Four line segments belonging to three different trajectories within one leaf node of an R-tree.

Let us consider for instance the popular TB-tree, in which a leaf node stores only contiguous line segments that belong to

the same trajectory and leaf nodes that store segments from the same trajectory are chained in a linked list. As a result, the TB-tree has high “temporal discrimination” (this terminology was introduced in [6]). Figure 1 shows a trajectory stored inside four leaf nodes within a TB-tree (each leaf node is shown as a bounding box). The curved and continuous appearance of the trajectory is because multiple line segments are stored together in each leaf node. By contrast, the R-tree simply stores in each leaf node trajectory segments that are spatially and temporally near each other, regardless of the individual trajectories. Figure 2 depicts an example with 4 segments belonging to 3 different trajectories that could be stored in a leaf node of an R-tree. For a distance threshold query, the number of TB-tree leaf nodes processed to perform the search could be arbitrarily high (since segment MBBs from many different trajectories can overlap the query MBB). Therefore, the TB-tree reduces the important R-tree property of overlap reduction; with an R-tree it may be sufficient to process only a few leaf nodes since each leaf node stores spatially close segments from multiple trajectories. For distance threshold queries, high spatial discrimination is likely to be more efficient than high temporal discrimination. Also, results in [6] show that the TB-tree performs better than the R-tree (for  $k$ NN queries) especially when the number of indexed entries is low; however, we are interested in large MODs (see Section III-A). We conclude that an R-tree index should be used for efficient distance threshold query processing.

## V. SEARCH ALGORITHM

We propose an algorithm, TRAJDISTSEARCH (Figure 3), to search for trajectories that are within a threshold distance of a query trajectory (defined as a set of connected trajectory segments over some temporal extent). All entry MBBs that overlap the query MBB are returned by the R-tree index and are then processed to determine the result set. More specifically, the algorithm takes as input an R-Tree index,  $T$ , a query trajectory,  $Q$ , and a threshold distance,  $d$ . It returns a set of time intervals annotated by trajectory ids, corresponding to the interval of time during which a particular trajectory is within distance  $d$  of the query trajectory. After initializing the result set to the empty set (line 2), the algorithm loops over all (augmented) MBBs that correspond to the segments of the query trajectory (line 3). For each such query MBB, the R-Tree index is searched to obtain a set of candidate entry MBBs that overlap the query MBB (line 4). The algorithm then loops over all the candidates (line 5) and does the following. First, given the candidate entry MBB and the query MBB, it computes an entry trajectory segment and a query trajectory segment that span the same time interval (line 6). The algorithm then computes the interval of time during which these two trajectory segments are within a distance  $d$  of each other (line 7). This calculation involves computing the coefficients of and solving a degree two polynomial [15]. If this interval is non-empty, then it is annotated with the trajectory id and added to the result set (line 9). The overall result set is returned once all query MBBs have been processed (line 13). Note that for a static point search  $Q.MBBSet$  (line 3) would consist of a single (degenerate) MBB with a zero extent in all spatial dimensions and some temporal extent, thus obviating the need for the outer loop. We call this algorithm POINTDISTSEARCH.

```

1: procedure TRAJDISTSEARCH (R-Tree  $T$ , Query  $Q$ , double  $d$ )
2:   resultSet  $\leftarrow \emptyset$ 
3:   for all querySegmentMBB in  $Q.MBBSet$  do
4:     CandidateSet  $\leftarrow T.Search(querySegmentMBB, d)$ 
5:     for all candidateMBB in CandidateSet do
6:       (EntrySegment, QuerySegment)  $\leftarrow interpolate($ 
           candidateMBB, querySegmentMBB)
7:       timeInterval  $\leftarrow calcTimeInterval($ 
           EntrySegment, QuerySegment,  $d)$ 
8:       if timeInterval  $\neq \emptyset$  then
9:         resultSet  $\leftarrow resultSet \cup timeInterval$ 
10:      end if
11:    end for
12:  end for
13:  return resultSet
14: end procedure

```

Figure 3. Pseudo-code for the TRAJDISTSEARCH algorithm (Section V).

## VI. INITIAL EXPERIMENTAL EVALUATION

### A. Datasets

Our first dataset, *Trucks* [18], is used in other MOD works [13], [14], [15]. It contains 276 trajectories corresponding to 50 trucks that travel in the Athens metropolitan area for 33 days. This is a 3-dimensional dataset (2 spatial + 1 temporal). Our second dataset is a class of 4-dimensional datasets (3 spatial + 1 temporal), *Galaxy*. These datasets contain the trajectories of stars moving in the Milky Way’s gravitational field (see Section III-A). The largest *Galaxy* dataset consists of 1,000,000 trajectory segments corresponding to 2,500 trajectories of 400 timesteps each. Distances are expressed in kiloparsecs (kpc). Our third dataset is a class of 4-dimensional synthetic datasets, *Random*, with trajectories generated via random walks. An adjustable parameter,  $\alpha$ , is used to control whether the trajectory is a straight line ( $\alpha = 0$ ) or a Brownian motion trajectory ( $\alpha = 1$ ). We vary  $\alpha$  in 0.1 increments to produce 11 datasets for datasets containing between  $\sim 1,000,000$  and  $\sim 5,000,000$  segments. Trajectories with  $\alpha = 0$  spans the largest spatial extent and trajectories with  $\alpha = 1$  are the most localized. All trajectories have the same temporal extent but different start times. Other synthetic datasets exist, such as GSTD [19]. We do not use GSTD because it does not allow for 3-dimensional spatial trajectories.

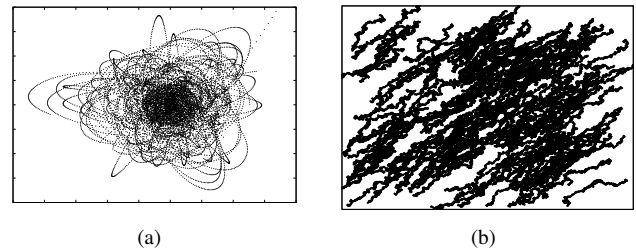


Figure 4. (a) *Galaxy* dataset: a sample of 30 trajectories, (b) 200 trajectories in the *Random* dataset with  $\alpha = 0.8$ .

Figure 4 shows a 2-D illustration of the *Galaxy* and *Random* datasets. An illustration of *Trucks* can be found in previous works [13], [14]. Table I summarizes the main characteristics of each dataset. The *Galaxy* and *Random* datasets are publicly available [20].

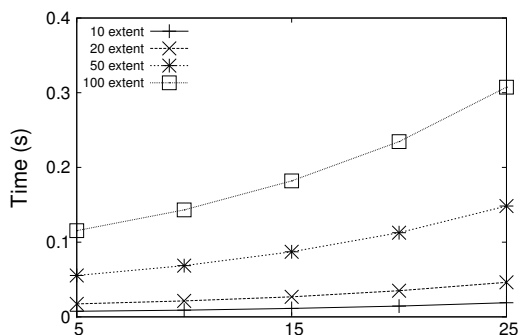
TABLE I. CHARACTERISTICS OF DATASETS

Dataset	Trajec.	Entries
Trucks	276	112152
Galaxy-200k	500	200000
Galaxy-400k	1000	400000
Galaxy-600k	1500	600000
Galaxy-800k	2000	800000
Galaxy-1M	2500	1000000
Random-1M ( $\alpha \in \{0, 0.1, \dots, 1\}$ )	2500	997500
Random-2M ( $\alpha = 1$ )	5000	1995000
Random-3M ( $\alpha = 1$ )	7500	2992500
Random-4M ( $\alpha = 1$ )	10000	3990000
Random-5M ( $\alpha = 1$ )	12500	4987500

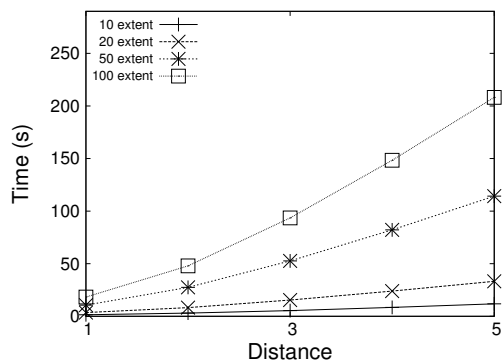
B. Experimental Methodology

We have implemented algorithm TRAJDISTSEARCH in C++, reusing an existing R-Tree implementation based on that initially developed by A. Guttman [5], and the code is available [21]. We execute this implementation on one core of a dedicated Intel Xeon X5660 processor, at 2.8 GHz, with 12 MB L3 cache and sufficient memory to store the entire index. We measure query response time averaged over 3 trials. The variation among the trials is negligible so that error bars in our results are not visible. We ignore the overhead of loading the R-Tree from disk into memory, which can be done once before all query processing.

C. Trajectory Search Performance



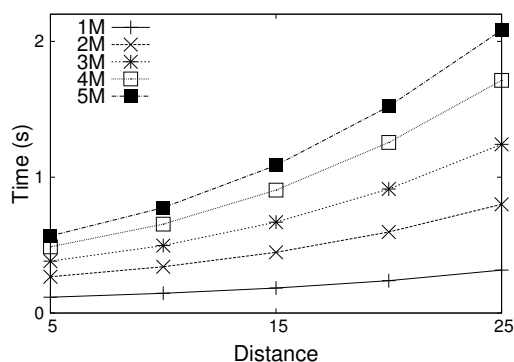
(a) Random  $\alpha = 1$



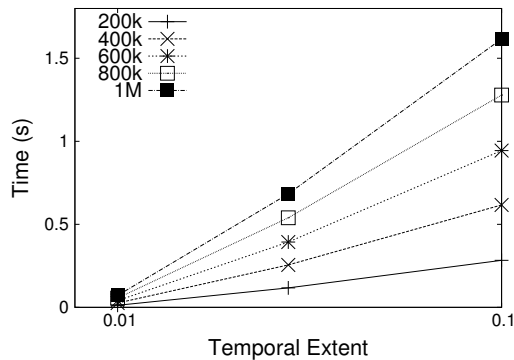
(b) Galaxy-1M

Figure 5. Query response time vs. threshold distance for 10%, 20%, 50% and 100% of the temporal extents of trajectories in S1 using the Random-1M  $\alpha = 1$  dataset (a) and the Galaxy-1M dataset with search S2 (b).

We measure the query response time of TRAJDISTSEARCH for the following sets of trajectory searches:



(a) Random  $\alpha = 1$



(b) Galaxy

Figure 6. (a) Response time vs. threshold distances for various numbers of segments in the index using search S3. (b) Response time vs. temporal extent for various numbers of segments in the index using search S4.

- S1: Random-1M dataset,  $\alpha = 1$ , 100 randomly selected query trajectories, processed for 10%, 20%, 50% and 100% of their temporal extents, with various query distances;
- S2: Same as S1 but for the Galaxy-1M dataset;
- S3: Random-1M, 2M, 3M, 4M and 5M datasets,  $\alpha = 1$ , 100 randomly selected query trajectories, processed for 100% of their temporal extent, with various query distances;
- S4: Galaxy-200k, 400k, 600k, 800k, 1M datasets, 100 randomly selected trajectories, processed with for 1%, 5% and 10% of their temporal extents, with a fixed query distance  $d = 1$ .

Figures 5 (a) and 5 (b) plot response time vs. query distance for S1 and S2 above. The response time increases slightly superlinearly with the query distance and with the temporal extents. In other words, the R-tree search performance degrades gracefully as the search is more extensive. Figures 6 (a) and (b) show response time vs. query distance for S3 and S4 above. The response time increases slightly superlinearly as the query distance increases for S3, and roughly linearly with the temporal extent increases for S4. Both these figures show results for various dataset sizes. An important observation is that the response time degrades gracefully as the datasets increase in size. More interestingly, note that for a fixed temporal extent and a fixed query distance, a larger dataset means a higher trajectory density, and thus a higher degree of overlap in the R-tree index. In spite of this increasing overlap,

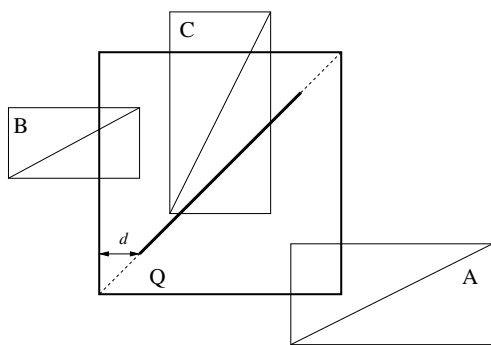


Figure 7. Three example entry MBBs and their overlap with a query MBB.

the R-tree still delivers good performance. We obtained similar results for POINTDISTSEARCH, which are omitted here.

## VII. TRAJECTORY SEGMENT FILTERING

The results in the previous section illustrate that TRAJDISTSEARCH maintains roughly consistent performance behavior over a range of query configurations (temporal extents, threshold distances, index sizes). In this and the next section, we explore approaches to reduce response time.

At each iteration our algorithm computes the moving distance between two line segments (line 7 in Figure 3). One can bypass this computation by “filtering out” those line segments for which it is straightforward (i.e., computationally cheap) to determine that they cannot possibly lie within distance  $d$  of the query. This filtering is applied to the segments once they have been returned by the index, and is thus independent of the indexing method.

Figure 7 shows an example with a query MBB,  $Q$ , and three overlapping MBBs,  $A$ ,  $B$ , and  $C$ , that have been returned from the index search. The query distance  $d$  is indicated in the (augmented) query box so that the query trajectory segment is shorter than the box’s diagonal.  $A$  contains a segment that is outside  $Q$  and should thus be filtered out. The line segment in  $B$  crosses the query box boundary but is never within distance  $d$  of the query segment and should be filtered out.  $C$  contains a line segment that is within a distance  $d$  of the query segment, and should thus not be filtered out. For this segment a moving distance computation must be performed (Figure 3, line 7) to determine whether there is an interval of time in which the two trajectories are indeed within a distance  $d$  of each other. The fact that candidate segments are returned that should in fact be ignored is inherent to the use of MBBs: a segment occupies an infinitesimal portion of its MBB’s space. This issue is germane to MODs that store trajectories using MBBs.

In practice, depending on the dataset and the search, the number of line segments that should be filtered out can be large. Figure 8 shows the number of candidate segments returned by the index search and the number of segments that are within the query distance vs.  $\alpha$ , for the *Random-1M* dataset, with 100 randomly selected query trajectories processed for 100% of their temporal extent. The fraction of candidate segments that are within the query distance is below 16.5% at  $\alpha = 1$ . In this particular example, an ideal filtering method would filter out more than 80% of the line segments.

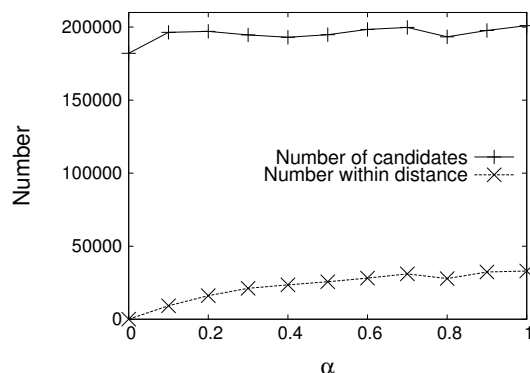


Figure 8. Total number of moving distance calculations vs. the number that are actually within a distance of 15 in the *Random-1M* datasets.

### A. Two Segment Filtering Methods

After the query and entry line segments are interpolated so that they have the same temporal extent (Figure 3, line 6), various criteria may remove the candidate segment from consideration. We consider two filtering methods beyond the simple no filtering approach:

**Method 1** – No filtering.

**Method 2** – After the interpolation, check whether the candidate segment still lies within the query MBB. This check only requires floating point comparisons between spatial coordinates of the segment endpoints and the query MBB corners, and would occur between lines 6 and 7 in Figure 3. Method 2 would filter out  $A$  in Figure 7.

**Method 3** – Considering only 2 spatial dimensions, say  $x$  and  $y$ , for a given query segment MBB compute the slope and the  $y$ -intercept of the line that contains the query segment. This computation requires only a few floating point operations and would occur in between lines 3 and 4 in Figure 3, i.e., in the outer loop. Then, before line 7, check if the endpoints of the candidate segment both lie more than a distance  $d$  above or below the query trajectory line. In this case, the candidate segment can be filtered out. This check requires only a few floating point operations involving segment endpoint coordinates and the computed slope and  $y$ -intercept of the query line. Method 3 would filter out both  $A$  and  $B$  in Figure 7.

Other computational geometry methods could be used for filtering, but these methods must be sufficiently fast (i.e., low floating point operation counts) if any benefit over Method 1 is to be achieved.

### B. Filtering Performance

We have implemented the filtering methods in the previous section in TRAJDISTSEARCH and in this section we measure response times ignoring the R-tree search, i.e., focusing only on the filtering and the moving distance computation. We use the following distance threshold searches:

- S5: From the *Trucks* dataset, 10 trajectories are processed for 100% of their temporal extent.
- S6: From the *Galaxy-1M* dataset, 100 trajectories are processed for 100% of their temporal extent.
- S7: From the *Random-1M* datasets, 100 trajectories are processed for 100% of their temporal extent, with a fixed query distance  $d = 15$ .

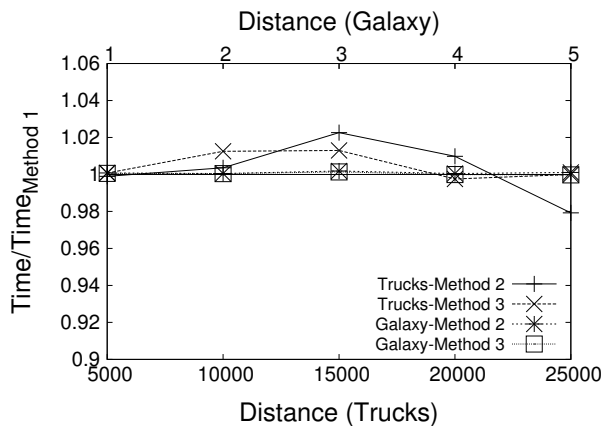


Figure 9. Performance improvement ratio of filtering methods for real datasets with S5 and S6, vs. query distance.

Figure 9 plots the relative improvement (i.e., ratio of response times) of using Method 2 and Method 3 over using Method 1 vs. the threshold distance for S5 and S6 above for the Galaxy and Trucks datasets. Data points below the  $y = 1$  line indicate that filtering is beneficial. We see that filtering is almost never beneficial and can in fact marginally increase response time. Similar results are obtained for the Random dataset regardless of the  $\alpha$  value.

It turns out that our methods filter only a small fraction of the line segments. For instance, for search S7 Method 2, resp. Method 3, filters out between 2.5% and 12%, resp. between 3.2% and 15.9%, of the line segments. Therefore, for most candidate segments the time spent doing filtering is pure overhead. Furthermore, filtering requires only a few floating point operations but also several if-then-else statements. The resulting branch instructions slow down executions (due to pipeline stalls) when compared to straight line code. We conclude that, at least for the datasets and searches we have used, our filtering methods are not effective.

One may envision developing better filtering methods to achieve (part of the) filtering potential seen in Figure 8. We profiled the execution of TRAJDISTSEARCH for searches S5, S6, and S7, with no filtering, and accounting both for the R-tree search and the distance computation. We found that the time spent searching the R-tree accounts for at least 97% of the overall response time. As a result, filtering can only lead to marginal performance improvements for the datasets and queries in our experiments. For other datasets and queries, however, the fraction of time spent computing distances could be larger. Nevertheless, given the results in this section, in all that follows we do not perform any filtering.

## VIII. INDEX RESOLUTION

According to the cost model in [22], index performance depends on the number of nodes in the index, but also on the volume and surface area of the MBBs. One extreme is to store an entire trajectory in a single MBB as defined by the spatial and temporal properties of the trajectory; however, this leads to a lot of “wasted MBB space.” Representing the object using multiple MBBs decreases the amount of empty space by storing the object in a series of consecutive multi-segment MBBs. The other extreme is to store each trajectory

line segment in its own MBB, as done so far in this paper and in previous work on  $k$ NN queries [12], [13], [14], [15]. In this scenario, the volume occupied by the trajectory in the index is minimized, with the trade-off that the number of entries in the index will be maximized.

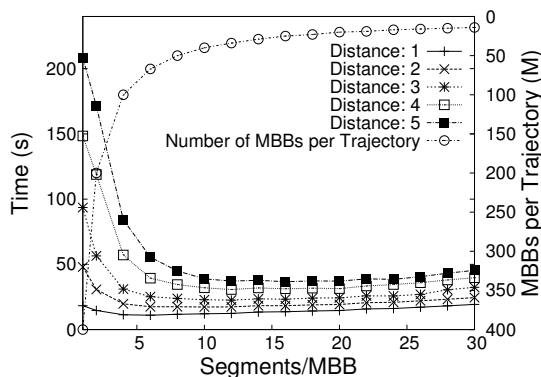
Assigning a fraction of a trajectory to a single MBB, as a series of line segments, increases overlap in the index, as the resulting MBB is larger in comparison to minimizing the volume of the MBBs by describing each individual trajectory line segment by its own MBB. As a result, an index search can return a portion of a trajectory that does not overlap the query, leading to increased overhead when processing the candidate set of line segments returned by the index. However, the number of entries in the index is reduced, thereby reducing tree traversal time. To explore the tradeoff between number of nodes in the index, the amount of wasted volume required by a trajectory, index overlap, and the overhead of processing candidate trajectory segments, in this section we evaluate three strategies for splitting trajectories into a series of consecutive MBBs, implemented as an array of references to trajectory segments (leading to one extra indirection when compared to assigning a single segment per MBB). We evaluate performance experimentally by splitting the trajectories, and then creating their associated indexes, where the configuration with the lowest query response time is highlighted. We leave analytical performance models of trajectory splitting methods for future work.

### A. Static Temporal Splitting

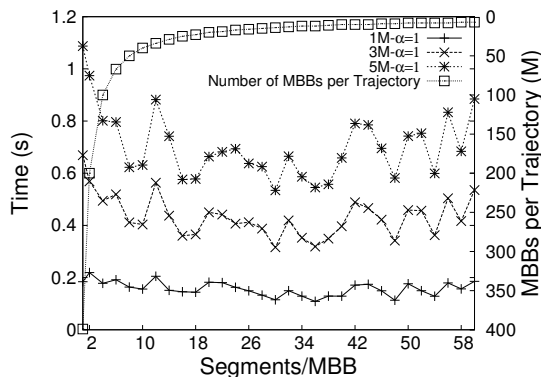
Assuming it is desirable to ensure that trajectory segments are stored contiguously, we propose a simple method. Given a trajectory of  $n$  line segments, we split the trajectory by assigning  $r$  contiguous line segments per MBB, where  $r$  is a constant. Therefore, the number of MBBs,  $M$  that represent a single trajectory is  $M = \lceil \frac{n}{r} \rceil$ . By storing segments contiguously, this strategy leads to high temporal locality of reference, which may be important for cache reuse in our in-memory database, in addition to the benefits of the high spatial discrimination of the R-tree (see Section IV).

Figure 10 plots response time vs.  $r$  for the S6 (*Galaxy* dataset) and S7 (*Random* dataset) searches defined in Section VII-B. For S6, 5 different query distances are used, while for S7 the query distance is fixed as 15 but results are shown for various dataset sizes for  $\alpha = 1$ . The right y-axis shows the number of MBBs used per trajectory. The data points at  $r = 1$  correspond to the original implementation (rather than the implementation with  $r = 1$ , which would include one unnecessary indirection).

The best value for  $r$  depends on the dataset and the search. For instance, in the *Galaxy*-1M dataset (S6) using 12 segments per MBB leads to the best performance (or  $M = 34$ ). We note that picking a  $r$  value in a large neighborhood around this best value would lead to only marginally higher query response times. In general, using a small value of  $r$  can lead to high response times, especially for  $r = 1$  (or  $M = 400$ ). For instance, for S6 with a query distance of 5, the response time with  $r = 1$  is above 208 s while it is just above 37 s with  $r = 12$ . With  $r = 1$  the index is large and thus time-consuming to search. A very large  $r$  value does not lead to the



(a) *Galaxy-1M*



(b) *Random datasets*

Figure 10. Static Temporal Splitting: Response time vs.  $r$  for (a) S6 for the *Galaxy-1M* dataset for various query distances; and (b) S7 for the *Random-1M*, 3M, and 5M  $\alpha = 1$  datasets and a query distance of 15. The number of MBBs per trajectory,  $M$ , is shown on the right vertical axis.

lowest response time since in this case many of the segments returned from the R-tree search are not query matches. Finally, results in Figure 10 (a) show that the advantage of assigning multiple trajectory segments per MBB increases as the query distance increases. For instance, for a distance of 2 using  $r = 12$  decreases the response time by a factor 2.76 when compared to using  $r = 1$ , while this factor is 5.6 for a distance of 5. Note that the difference in response times between Figure 10 (a) and (b) are largely due to more queries within  $d$  in *Galaxy* in comparison to *Random* for the query distances selected.

### B. Static Spatial Splitting

Another strategy consists in ordering the line segments belonging to a trajectory spatially, i.e., by sorting the line segments of a trajectory by the  $x$ ,  $y$ , and  $z$  values of the segment's origin. We then assign  $r$  segments per trajectory into each MBB, as in the previous method. With such spatial grouping, the line segments are no longer guaranteed to be temporally contiguous in their MBBs, but reduced index overlap may be achieved. Figure 11 plots response time vs.  $r$  for the S7 (*Random* dataset) searches. We see that there is no advantage to assigning multiple trajectory segments to an MBB over assigning a single line segment to a MBB ( $r = 1$  in the plot). When comparing with results in Figure 10 (b) we find that spatial splitting leads to query response times higher by several factors than that of temporal splitting.

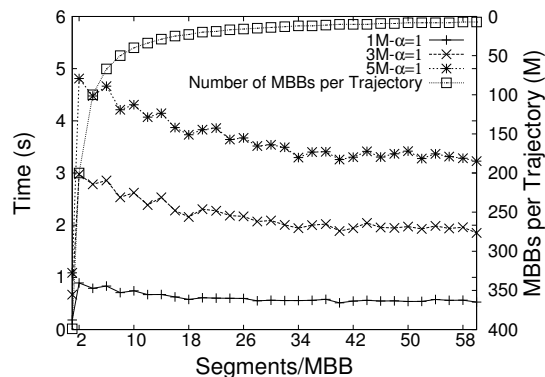


Figure 11. Static Spatial Splitting: Response time vs.  $r$  using S7 for the *Random-1M*, 3M, and 5M  $\alpha = 1$  datasets and a query distance of 15. The number of MBBs per trajectory,  $M$ , for each data point is shown on the rightmost vertical axis.

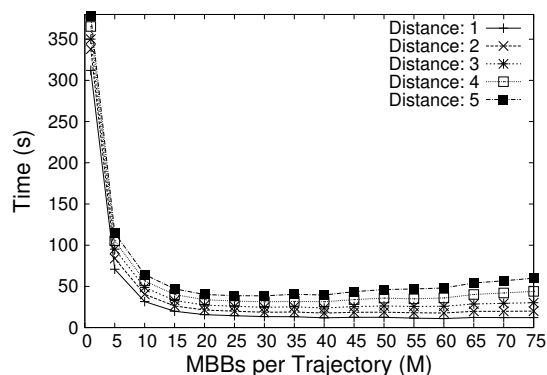
### C. Splitting to Reduce Trajectory Volume

The encouraging results in Section VIII-A suggest that using an appropriate trajectory splitting strategy can lead to performance gains primarily by exploiting the trade-off between the number of entries in the index and the amount of wasted space that leads to higher index overlap. More sophisticated methods can be used. In particular, we implement the heuristic algorithm *MergeSplit* in [23], which is shown to produce a splitting close to optimal in terms of wasted space. *MergeSplit* takes as input a trajectory,  $T$ , as a series of  $l$  line segments, and a constant number of MBBs,  $M$ . As output, the algorithm creates a set of  $M$  MBBs that encapsulate the  $l$  segments of  $T$ . The pseudocode of *MergeSplit* is as follows:

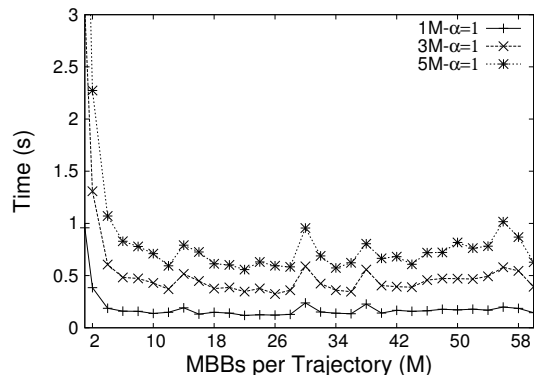
- 1) For  $0 \leq i < l$  calculate the volume of the merger of the MBBs that define  $l_i$  and  $l_{i+1}$  and store the resulting series of MBBs and their volumes.
- 2) To obtain  $M$  MBBs, repeat  $(l-1)-(M-1)$  times and merge consecutive MBBs that produce the smallest volume increase at each step. After the first iteration, there will be  $l - 2$  initial MBBs describing line segments, and one MBB that is the merger of two line segment MBBs.

Figure 12 shows response time vs.  $M$  for S6 (*Galaxy* dataset) and S7 (*Random* dataset). Compared to static temporal splitting, which has a constant number of segments,  $r$  per MBB, *MergeSplit* has a variable number of segments per MBB. From the figure, we observe that for the *Galaxy-1M* dataset (S6),  $M = 30$  leads to the best performance. Comparing *MergeSplit* to the static temporal splitting (Figures 10 and 12 (a)), the best performance for the S6 (*Galaxy* dataset) is achieved by the static temporal splitting. For S7, the *Random-1M*, 3M, and 5M  $\alpha = 1$  datasets, *MergeSplit* is only marginally better than the static temporal splitting (Figures 10 and 12 (b)). This is surprising, given that the total hypervolume of the entries in the index for a given  $M$  across both splitting strategies is higher for the simple static temporal splitting, as it makes no attempt to minimize volume. Therefore, the trade-off between the number of entries and overlap in the index cannot fully explain the performance of these trajectory splitting strategies for distance threshold queries.





(a) *Galaxy-1M*

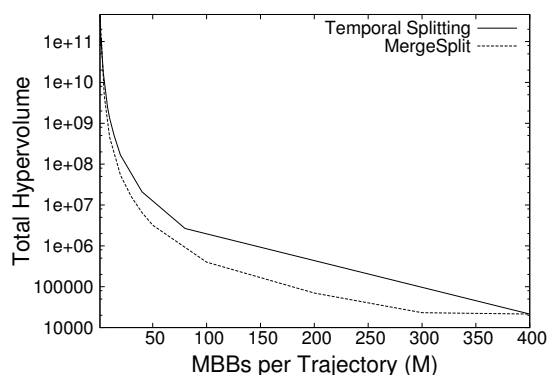


(b) *Random datasets*

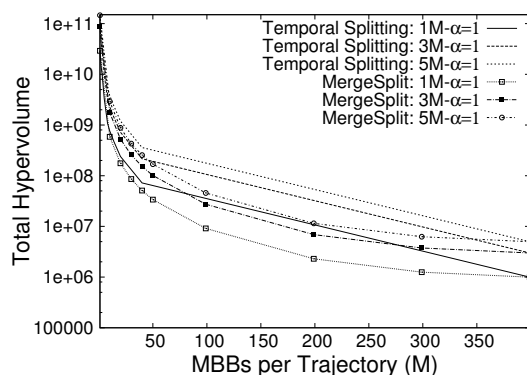
Figure 12. Greedy Trajectory Splitting: Response time vs.  $M$  for (a) S6 for the *Galaxy-1M* dataset for various query distances; and (b) S7 for the *Random-1M*, 3M, and 5M  $\alpha = 1$  datasets and a query distance of 15.

#### D. Discussion

A good trade-off between the number of entries in the index and the amount of index overlap can be achieved by selecting an appropriate trajectory splitting strategy. However, comparing the results of the simple temporal splitting strategy (Section VIII-A) and *MergeSplit* (Section VIII-C), we find that volume minimization did not significantly improve performance for S7, and led to worse performance for S6. In Figure 13, we plot the total hypervolume vs.  $M$  for the *Galaxy-1M* (S6) and the *Random-1M*, 3M, and 5M  $\alpha = 1$  (S7) datasets.  $M = 1$  refers to placing an entire trajectory in a single MBB, and the maximum value of  $M$  refers to placing each individual line segment of a trajectory in its own MBB. For the static temporal splitting strategy,  $M = 34$  leads to the best performance for the *Galaxy-1M* dataset (S6), whereas this value is  $M = 30$  for *MergeSplit*. The total hypervolume of the MBBs in units of  $\text{kpc}^3\text{Gyr}$  for the static temporal grouping strategy at  $M = 34$  is  $3.6 \times 10^7$ , whereas for *MergeSplit* at  $M = 30$ , it is  $1.62 \times 10^7$ , or the MBBs require 55% less volume. Due to the greater volume occupied by the MBBs, index overlap is much higher for the static temporal splitting strategy. Figure 14 (a) plots the number of overlapping line segments vs.  $M$  for S6 with  $d = 5$ . From the figure, we observe that independently of  $M$ , *MergeSplit* returns a greater number of candidate line segments to process than the simple temporal splitting strategy. *MergeSplit* attempts to minimize volume; however, if an MBB contains a significant fraction of the line segments of a given trajectory, then all of these



(a) *Galaxy-1M*



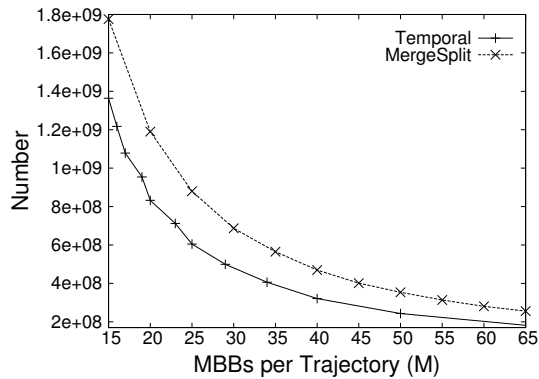
(b) *Random-1M*

Figure 13. Total hypervolume vs.  $M$  for the static temporal splitting strategy and *MergeSplit*. (a) for the *Galaxy-1M* dataset (S6); and (b) for the *Random-1M*, 3M, and 5M  $\alpha = 1$  datasets (S7).

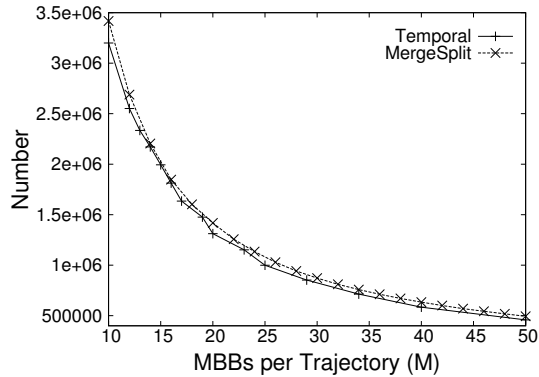
segments are returned as candidates. The simple temporal grouping strategy has an upper bound ( $r$ ) on the number of segments returned per overlapping MBB and thus can return fewer candidate segments for a query, despite occupying more volume in the index. For in-memory distance threshold queries, there is a trade-off between a trajectory splitting strategy that has an upper bound on the number of line segments per MBB, and index overlap, characterized by the volume occupied by the MBBs in the index. This is in sharp contrast to other works that focus on efficient indexing of spatiotemporal objects in traditional out-of-core implementations where the index resides partially in-memory and on disk, and therefore volume reduction to minimize index overlap is necessary to minimize disk accesses (e.g., [23]).

#### E. Performance Considerations for In-memory and Out-of-Core Implementations

The focus of this work is on in-memory distance threshold queries; however, most of the literature on MODs assume out-of-core implementations, where the number of node accesses are used as a metric to estimate I/O activity. Figure 15 shows the number of node accesses vs.  $M$  for both of the static temporal splitting strategy and *MergeSplit*. We find that for the *Galaxy-1M* dataset (S6) with  $d = 5$ , there are a comparable number of node accesses for both trajectory splitting methods. However, for S7 (*Random-1M*), on average, trajectory splitting with *MergeSplit* requires fewer node accesses and may perform significantly better than the simple temporal splitting

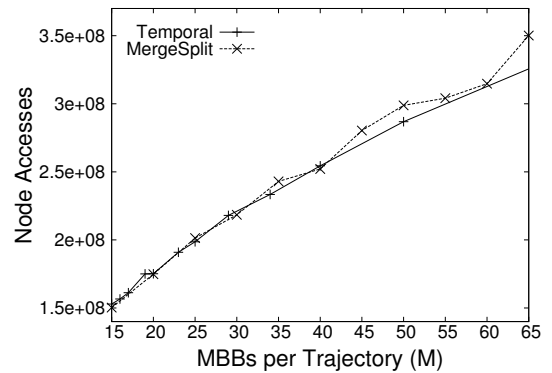


(a) *Galaxy-1M*

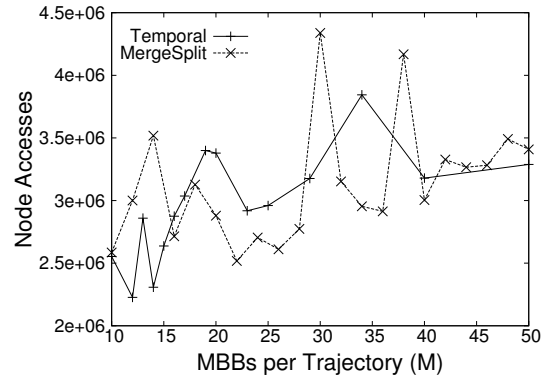


(b) *Random-1M*

Figure 14. Total number of overlapping segments vs.  $M$  for the static temporal splitting strategy and *MergeSplit*. (a) S6 for the *Galaxy-1M* dataset with  $d = 5$ ; and (b) S7 for the *Random*  $\alpha = 1$  dataset with  $d = 15$ .



(a) *Galaxy-1M*



(b) *Random-1M*

Figure 15. Node Accesses vs.  $M$  for the static temporal splitting strategy and *MergeSplit*. (a) S6 for the *Galaxy-1M* dataset with  $d = 5$ ; and (b) S7 for the *Random*  $\alpha = 1$  dataset with  $d = 15$ .

strategy in an out-of-core implementation. For example, in Figure 15 (b) some values of  $M$  have a significantly higher number of node accesses, such as values around 14, 30, 38, due to the idiosyncrasies of the data, and resulting index overlap. However, as we demonstrated in Section VIII-D, distance threshold queries in the context of in-memory databases also benefit from reducing the number of candidate line segments returned, and this is not entirely volume contingent. Therefore, methods that consider volume reduction, such as the *Merge-Split* algorithm of [23], or other works that consider volume reduction in the context of query sizes, such as [24], may not be entirely applicable to distance threshold queries.

A single metric cannot capture the trade-offs between the number of entries in the index, volume reduction, index overlap, and the number of candidate line segments returned (germane to distance threshold queries). However, for *Galaxy-1M* (S6), a value of  $M = 34$  and  $M = 30$  lead to the best query response time for the temporal splitting strategy and *MergeSplit*, respectively (Figures 10 (a) and 12 (a)). Figure 16 shows the number of L1 cache misses vs.  $M$  for S6 with  $d = 5$ . The number of cache misses was measured using PAPI [25]. The best values of  $M$  in terms of query response time for both of the trajectory splitting strategies ( $M = 34$  and  $M = 30$ ) roughly correspond to a value of  $M$  that minimizes cache misses. Thus, cache misses appear to be a good indicator of relative query performance under different indexing methods. Future work for in-memory distance threshold queries should focus on improved cache reuse through temporal locality of

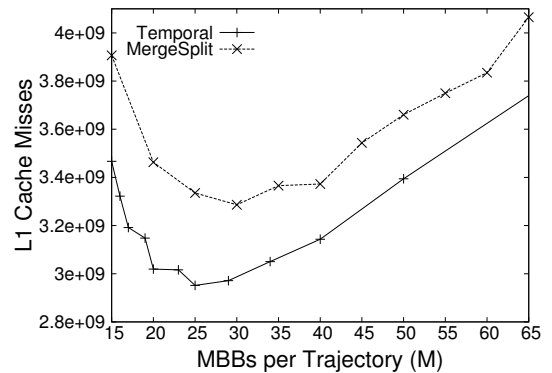


Figure 16. L1 cache misses vs.  $M$  for the static temporal splitting strategy and *MergeSplit* for the *Galaxy-1M* dataset (S6) with  $d = 5$ .

reference (which is in part obtained by storing segments contiguously within a single MBB).

## IX. CONCLUSION

In-memory distance threshold queries for trajectory and point queries on moving object trajectories are significantly different from the well-studied  $k$ NN searches [12], [13], [14], [15]. We made a case for using an R-tree index to store trajectory segments, and found it to perform robustly for two real world datasets and a synthetic dataset. We focused on 4-D datasets (3 spatial + 1 temporal) while other works only consider 3-D datasets [12], [13], [14], [15].

We found the popular “search and refine” strategy to be ineffective for distance threshold searches since many segments returned by the index search must be excluded from the result set. We have proposed computationally inexpensive solutions to filter out candidate segments, but found that they have poor selectivity. A more promising direction for reducing query response time is to reduce the time spent traversing the tree index. We demonstrated that efficiently splitting trajectories is beneficial because the penalty for the increased index overlap is offset by the reduction in number of index entries. We find that for in-memory distance threshold queries, the number of line segments returned per overlapping MBB has an impact on performance, where attempts to reduce the volume of the MBBs that store a trajectory may be at cross-purposes with returning a limited number of candidate segments per overlapping MBB. Therefore, trajectory splitting methods that focus on volume reduction are not necessarily preferable to a simple and bounded grouping of line segments in MBBs.

A future direction is to explore trajectory splitting methods that achieve volume reduction while bounding the number of MBBs used per trajectory. Another direction is to investigate non-MBB-based data structures to index line segments, such as that in [26]. Finally, we plan to develop algorithms for parallel processing of in-memory distance threshold queries both for shared- and distributed-memory executions.

One may wonder whether the idea of assigning multiple segments to an MBB is generally applicable, and in particular for  $k$ NN searches on trajectories [12], [13], [14], [15]. The  $k$ NN literature focuses on pruning strategies and associated metrics that require a high resolution index, thus implying storing a single trajectory segment in an MBB. Furthermore,  $k$ NN query processing algorithms maintain a list of nearest neighbors over a time interval, which would lead to greater overhead if multiple segments were stored per MBB. Therefore, the approach of grouping line segments together in a single MBB may be ineffective for  $k$ NN queries. An interesting problem is to reconcile the differences between both types of queries in terms of index resolution.

#### ACKNOWLEDGMENTS

This paper has benefited from the insightful comments of Lipeow Lim. This material is based upon work supported by the National Aeronautics and Space Administration through the NASA Astrobiology Institute under Cooperative Agreement No. NNA08DA77A issued through the Office of Space Science, and by NSF Award CNS-0855245.

#### REFERENCES

- [1] L. Forlizzi, R. H. Güting, E. Nardelli, and M. Schneider, “A data model and data structures for moving objects databases,” in Proc. of ACM SIGMOD Intl. Conf. on Management of Data, 2000, pp. 319–330.
- [2] R. H. Güting, M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis, “A foundation for representing and querying moving objects,” *ACM Trans. Database Syst.*, vol. 25, no. 1, 2000, pp. 1–42.
- [3] S. Arumugam and C. Jermaine, “Closest-point-of-approach join for moving object histories,” in Proc. of the 22nd Intl. Conf. on Data Engineering, 2006, pp. 86–95.
- [4] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen, “Discovery of convoys in trajectory databases,” *Proc. VLDB Endow.*, vol. 1, no. 1, Aug. 2008, pp. 1068–1080.
- [5] A. Guttman, “R-trees: a dynamic index structure for spatial searching,” in Proc. of ACM SIGMOD Intl. Conf. on Management of Data, 1984, pp. 47–57.
- [6] D. Pfoser, C. S. Jensen, and Y. Theodoridis, “Novel Approaches in Query Proc. for Moving Object Trajectories,” in Proc. of the 26th Intl. Conf. on Very Large Data Bases, 2000, pp. 395–406.
- [7] Y. Theodoridis, M. Vazirgiannis, and T. Sellis, “Spatio-Temporal Indexing for Large Multimedia Applications,” in Proc. of the Intl. Conf. on Multimedia Computing and Systems, 1996, pp. 441–448.
- [8] V. P. Chakka, A. Everspaugh, and J. M. Patel, “Indexing large trajectory data sets with SETI,” in Proc. of Conference on Innovative Data Systems Research, 2003, pp. 164–175.
- [9] P. Cudre-Mauroux, E. Wu, and S. Madden, “TrajStore: An Adaptive Storage System for Very Large Trajectory Data Sets,” in Proc. of the 26th Intl. Conf. on Data Engineering, 2010, pp. 109–120.
- [10] M. R. Vieira, P. Bakalov, and V. J. Tsotras, “On-line discovery of flock patterns in spatio-temporal data,” in Proc. of the 17th ACM SIGSPATIAL Intl. Conf. on Advances in Geographic Information Systems, 2009, pp. 286–295.
- [11] Z. Li, M. Ji, J.-G. Lee, L.-A. Tang, Y. Yu, J. Han, and R. Kays, “Movemine: Mining moving object databases,” in Proc. of the 2010 ACM SIGMOD Intl. Conf. on Management of Data, 2010, pp. 1203–1206.
- [12] E. Frenzos, K. Gratsias, N. Pelekis, and Y. Theodoridis, “Nearest neighbor search on moving object trajectories,” in Proc. of the 9th Intl. Conf. on Advances in Spatial and Temporal Databases, 2005, pp. 328–345.
- [13] E. Frenzos, K. Gratsias, N. Pelekis, and Y. Theodoridis, “Algorithms for Nearest Neighbor Search on Moving Object Trajectories,” *Geoinformatica*, vol. 11, no. 2, 2007, pp. 159–193.
- [14] Y.-J. Gao, C. Li, G.-C. Chen, L. Chen, X.-T. Jiang, and C. Chen, “Efficient k-Nearest-Neighbor Search Algorithms for Historical Moving Object Trajectories,” *J. Comput. Sci. Technol.*, vol. 22, no. 2, 2007, pp. 232–244.
- [15] R. H. Güting, T. Behr, and J. Xu, “Efficient k-nearest neighbor search on moving object trajectories,” *The VLDB Journal*, vol. 19, no. 5, 2010, pp. 687–714.
- [16] M. G. Gowanlock, D. R. Patton, and S. M. McConnell, “A Model of Habitability Within the Milky Way Galaxy,” *Astrobiology*, vol. 11, 2011, pp. 855–873.
- [17] N. Roussopoulos, S. Kelley, and F. Vincent, “Nearest neighbor queries,” in Proc. of ACM SIGMOD Intl. Conf. on Management of Data, 1995, pp. 71–79.
- [18] <http://www.chorochronos.org/>, accessed 5-February-2014.
- [19] Y. Theodoridis, J. R. O. Silva, and M. A. Nascimento, “On the Generation of Spatiotemporal Datasets,” in Proc. of the 6th Intl. Symp. on Advances in Spatial Databases, 1999, pp. 147–164.
- [20] <http://navet.ics.hawaii.edu/%7Emike/datasets/DBKDA2014/datasets.zip>, accessed 12-February-2014.
- [21] <http://www.superliminal.com/sources/sources.htm>, accessed 5-February-2014.
- [22] B.-U. Pagel, H.-W. Six, H. Toben, and P. Widmayer, “Towards an analysis of range query performance in spatial data structures,” in Proc. of the 12th Symp. on Principles of Database Sys., 1993, pp. 214–221.
- [23] M. Hadjieleftheriou, G. Kollios, V. J. Tsotras, and D. Gunopulos, “Efficient indexing of spatiotemporal objects,” in Proc. of the 8th Intl. Conf. on Extending Database Technology: Advances in Database Technology, 2002, pp. 251–268.
- [24] S. Rasetic, J. Sander, J. Elding, and M. A. Nascimento, “A trajectory splitting model for efficient spatio-temporal indexing,” in Proc. of the 31st Intl. Conf. on Very Large Data Bases, 2005, pp. 934–945.
- [25] P. J. Mucci, S. Browne, C. Deane, and G. Ho, “PAPI: A Portable Interface to Hardware Performance Counters,” in Proc. of the Department of Defense HPCMP Users Group Conf., 1999, pp. 7–10.
- [26] E. Bertino, B. Catania, and B. Shidlovsky, “Towards Optimal Indexing for Segment Databases,” in Proc. of the 6th Intl. Conf. on Advances in Database Technology, 1998, pp. 39–53.

# Exploiting the Social Structure of Online Media to Face Transient Heavy Workload

Ibrahima Gueye and Idrissa Sarr  
 LID laboratory  
 Université Cheikh Anta Diop  
 Dakar, Senegal  
 (ibrahima82.gueye, idrissa.sarr)@ucad.edu.sn

Hubert Naacke  
 LIP6 Laboratory  
 Sorbonne Universités, UPMC Univ Paris 06  
 Paris, France  
 hubert.naacke@lip6.fr

**Abstract**—A challenging issue many online social media have to deal with is facing egocentric workloads that are very frequent. Such a situation is generally due to the simultaneous access of several users to a small piece of data owned by a user or a few ones. A key example is the number of comments posted on the Manchester United Facebook page after the manager announced his retirement (more than 1 billion comments on the related subjects). Since egocentric workloads are transient, two dimensions must be taken into account to deal with them: (1) the rapidity to react to the peak load and, (2) the lightness of the solution or its low cost. Therefore, the first goal of this paper is to exploit the underlying social structure of online social media to localize from which the peaks take place and to face them in their early stage. The second goal is to combine an elastic approach with a load balancing process to sustain the overall performances while minimizing the required resources. Our solution is evaluated through simulation with SimJava. The obtained results show the soundness of the approach as well as its feasibility.

**Keywords**—Transaction, Social workload, Load balancing, Elasticity.

## I. INTRODUCTION

Social media applications are characterized by online collaborative actions such as chatting, tagging and content sharing. The user experience is more and more guided by her social context or social position, i.e., a user with many connections tends to be involved in frequent online interactions. As reported in [1], the data belonging to the most popular users are the most frequently accessed. Furthermore, when a popular user acts in response to another user's action, this can cause other users to respond subsequently, generating a so called net effect. As a result, users may simultaneously access the same piece of data for a short period of time. We say that we face a set of *egocentric workloads* that are characterized by a socially dependent and fluctuating access pattern. The reason is that the overall workload derives from few users and their close contacts based on the status or role of users. To face egocentric workloads, a challenging issue is to deliver fast, scalable and cheap data access, using a reduced amount of resources.

### A. Motivations and Problem Statement

The interactions between users as well as the actions (comment, tag, etc.) made by a user on the items owned by others shape the well-known social structure. This structure is generally represented as a graph of a set of vertices with edges between them. Vertices are users or their items while edges are interactions or links between users. The number of neighbors or edges of a user is called *centrality degree*. A node with

a high number of neighbors is called a popular or important node and has therefore a high centrality degree value. Less important nodes are called peripheral nodes. Figure 1 depicts a social network where big rings represent popular users and small rings designate peripheral users.

It is obvious that popular users are involved more frequently than peripheral ones in online interactions. That is, paramount of the workload derives essentially from popular users and is the main reason we characterize the social workload as a set of egocentric workloads. Furthermore, an egocentric workload is transient since users behaviors are event-dependent and old events attract less attention leading to a disappearance of the related workload.

However, based on the interactions of users or their similarity, nodes can form groups for which the network connections are dense, but between which they are sparse. Such groups are called communities [2][3] or circles as in Google+. For instance, Figure 1 shows different groups: users within a group have a similar color. Moreover, users interact more with their neighbors within a circle than with others belonging to another circle. Thus, it is worth-noting that the overall workload is biased since the social position of a user as well as the size of its group impacts the number of interactions within the circle. Whatsoever the particularity of the social workload, it is made of by read and write intensive operations since (1) the overall number of users is very important and (2) almost every user action causes data read, insert and update. That is, even though the actions or interactions done by users are socially dependent, the generated workload is quite the same as the workload of classical applications (i.e., set of read and write operations). Therefore, egocentric peak load observed from social applications can be handled by using and adapting traditional techniques such as data partition and replication. The main issue to address therefore is how such techniques can be used for facing peak load while including social features. To face this issue, three problems may be formulated as follows

- How to detect data causing transient workload in its early stage within a social network?
- How to partition such data while ensuring fast interactions between a user and its contacts?
- How to forecast data that will cause a peak and to anticipate it based on the social structure?

Here are the set of problems we unveil and that we want to deal with through this paper. It is worth noting that even though the peak load is transient, it lasts thousand times longer



Figure 1: A social network graph.

than the time to execute a transaction. That is, partition and replication done for facing a peak load are cost for value.

### B. Contributions and paper organization

Our goal in this paper is to face the previous problems and the key novelties of our approach can be summarized as follows.

- A fine-grain identification of a peak load. Actually, we propose 1) a naive approach that relies on the number of transactions accessing a partition, and, 2) a social-based approach that uses interactions between users. The last approach is in fact a peak prediction model and it is designed using the homophily principle, which states that the flow of information from person to person is a declining function of distance in Blau space [4]. That is, it is possible to locate the scope of interactions initiated by a user, and to assess whether such interactions may lead to a peak. Moreover, since social network is composed by a set of communities, peak origins are located in those communities. Such a mechanism has the edge to isolate the peak origin and to face it locally.
- A lightweight data migration method that moves only relevant data, on a pull-on-demand basis, with minimal disruption on transactions processing. The data migration method is coupled with an elastic load balancing mechanism that is optimized for reducing resource usage while maintaining bounded response time.

The rest of this paper is structured as follows: in Section II, we present the the social workloads, basic concepts and global architecture of our system. In Section III, we show how we detect peak load as well as the prediction model we use to anticipate their appearance. In Section IV, we present the management of transient heavy workload. In Section V, we present the validation of our approaches and we highlight, in Section VI, a few related works before we conclude in Section VII.

## II. BASIC CONCEPTS AND GLOBAL ARCHITECTURE

In this section, we describe the global architecture we use and the social workloads we plan to face.

### A. Social workload

The workload is made of user actions. A user action is a sequence of transactions and we assume that each transaction reads and writes data owned by a single user. A user may share data with other users and grants consequently read and/or write permissions on them.

Actually, with a social networking website as Facebook or Google+, users have a various sort of data that may concern distinct subsets of their contacts. In other words, the user belongs to several circles. For instance, users may have professional circles that contain their items related to their professional activities and that will attract more their colleagues and collaborators. They can share a private circle with only their close friends and relatives. Therefore, the items of one user may be seen as a set of cohesive data that are more attached to a specific circle.

Furthermore, the workload looms from users with various popularity levels. Thus, a peak load may be observed on so called popular data belonging to popular users. Since all popular users are not active at the same time, therefore, the overall workload is not distributed uniformly over circles as well as over popular users (say we face a non-uniform distribution of the workload). In other words, the workload of the group  $i$  can be light while the one of the group  $j$  is heavy. With this insight, it is trivial to identify groups with peaks or those underloaded. Getting this kind of information has the edge to apply a selective mechanism to face peak load within a group while minimizing the cost and required time.

### B. Architecture

We devise an architecture using two layers: the routing layer and the datastore layer (see Figure 2). Our solution is a middleware that serves as an interface with the data manipulation procedures of applications. The routing layer is made of a set of nodes called client nodes (CN) and routers while the datastore layer contains database nodes (DB) that store data and execute queries. Data are stored on DB nodes by using community or cricle configuration in such a way that all related data of one group are on the same DB node. This is possible since the number of users within a circle is generally limited and hence, the related data can be hold in one single DB node. Transactions are sent by CN to any router, which afterwards forwards them to the right DB based on their access classes for execution. Note that each router stores a part of the global index, which allows them to locate data among database nodes. Transactions accessing the same data are routed in a serial way and the DBs guarantee consistent execution of transactions without locking.

Moreover, the routing layer includes a special and useful node called *Controller node (CtlN)*. It monitors the database layer for detecting whether a DB becomes a bottleneck or tends to be underloaded. In this respect, every DB sends periodically its load to the CtlN in order to permit overload detection based on a threshold. We mention that once a DB is found as overloaded, a migration process consisting of moving part of its data to a less loaded DB or a new one is initialized.

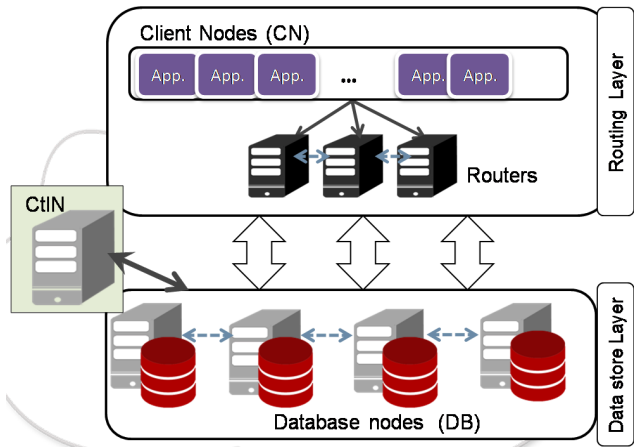


Figure 2: System architecture.

Reversely, an underloaded DB will bring out the merge of its data with another DB that has not enough load.

Furthermore, DB nodes are able to communicate between them for ensuring data migration from one to another in a consistent manner. To figure out the part of data to migrate from one to another DB, we rely on metadata hold via a data structure called *trace*. The *trace* records information about transactions such as their identifiers, their arrival dates, their waiting times.

### III. PEAK LOAD DETECTION

#### A. Definitions

We consider a set of nodes  $\mathcal{N}$ . Each node  $N_k \in \mathcal{N}$  is a (virtual) machine managing the database  $DB_k$ . Each  $DB_k$  node stores a set of partitions,  $p_i^k$  denoting the partition  $i$  of database  $k$ . During operation, a DB node executes the incoming transactions in sequence. Let  $\omega$  denote the most recent observation window, expressed in second. Each DB node logs the incoming transactions requests:  $T_\omega(p_i^k)$  denotes the set of transactions requesting the partition  $p_i^k$ , which either terminated during  $\omega$  or are not currently terminated (i.e., pending or running transactions). The log informs about the current execution time and waiting time of each transaction. To quantify the node load, we aggregate recent log information, let  $RT_\omega(p_i^k)$  denote sum of the execution and waiting times of all transactions in  $T_\omega(p_i^k)$ .

We define  $load(p_i^k)$  as the mean load of  $p_i^k$  within  $\omega$  as:

$$load(p_i^k) = \frac{RT_\omega(p_i^k)}{|\omega|} \quad (1)$$

Since a DB node may store many partitions based on its storage capacity, we define the load of a DB node as the sum of the loads of all partitions under its control. Formally, the load of a  $DB_k$  holding  $n$  partition is:

$$load(DB_k) = \sum_i load(p_i^k) \mid p_i^k \in DB_k \quad (2)$$

Let  $\tau_k$  be the standalone transaction processing time at node  $DB_k$ . Let  $rt_k$  be the observed transaction response

time (including the waiting time). The  $load(DB_k)$  can be considered as a penalty factor impacting  $rt_k$  as follows:  
 $rt_k = \tau_k \cdot load(DB_k)$

#### B. Detecting peak load

We define the stability conditions of every DB node as the conditions under which it is neither overloaded nor underloaded. More precisely, we expect every transaction to be executed in bounded time. Let  $T_{max}$  denote the maximum expected response time of a transaction. For each DB node, we expect  $rt_k \leq T_{max}$ , that is, the following condition must hold:

$$load(DB_k) \leq \frac{T_{max}}{\tau_k} \quad (3)$$

Reversely, a node is considered under-loaded if it remains idle (i.e., no transaction execution). Thus any DB node must satisfy the following condition:

$$load(DB_k) > 0 \quad (4)$$

A node is detected as overloaded (resp. idle) if the condition (3) (resp. (4)) does not hold for a given amount of time  $\omega_{overload}$  (resp.  $\omega_{idle}$ ). It is worth noting that  $T_{max}$  as well as the size of the time windows  $\omega_{overload}$  and  $\omega_{idle}$ , are key performance indicators.  $T_{max}$  can be set based on the SLA of the of cloud provider, while  $\omega$  values are tuned in order to make accurate decisions.

#### C. Identifying peak origins

A peak load occurs at a DB node if one or many partitions are overloaded, resulting in slow response time. With this respect, finding the origins of a peak can be summarized intuitively as identifying the sufficient set of partitions, with the highest load, that correspond to the extra load  $\Delta_{load}$  defined as:

$$\Delta_{load_k} = load(DB_k) - \frac{T_{max}}{\tau_k} \quad (5)$$

For each overloaded  $DB_k$  node, we sort its set of partitions  $\{p_i^k\}$  in descending order of  $load(p_i^k)$ . Then, we determine a subset  $M_k$  of  $\{p_i^k\}$  such that:

$$\sum_{p_i^k \in M_k} load(p_i^k) \geq \Delta_{load_k} \quad (6)$$

Notice that the size of  $M_k$  is minimal since  $M_k$  is a prefix of the ordered set  $\{p_i^k\}$ . This aim to further reduce the number of partitions to move. Next, we will move iteratively all partitions in  $M_k$ , to restore  $DB_k$  in a normal load status.

#### D. Predicting peak origins

The peak origin is identified previously based on the load of the partition. That is, we gather some statistics during a set of time windows before being able to detect peak. Briefly, we detect peak after their arrivals. However, we should be able to detect whether a pic will arrive soon based on the social interactions. We aim in this section to predict when a peak can happen based on data social characteristics or the social graph.

In fact, since each interaction corresponds to a transaction, thus, it is possible to estimate the number of transactions that are related to a user  $u_i$  and his/her related data. Therefore,

the partition that stores data of  $u_i$  can be continually checked whether it is overloaded or not.

To reach our goal, we rely entirely on the homophily principle, which states that the flow of information from person to person is a declining function of distance. Thus, using the degree centrality of nodes or its neighborhood can help to locate the scope of interactions initiated by a user, and to assess whether such interactions may lead to a peak.

We start from the point that each neighbor of  $u_i$  may either partake to  $u_i$ 's activities or not. Let  $p_m$  be the probability that  $m$  neighbors are involved in a given activity  $a_i$  of  $u_i$ . Thus, the number of transactions associated with  $a_i$  is approximately equal to  $m$  when considering that each neighbor of  $u_i$  reacts only one time to  $a_i$ . Therefore, it is trivial to express the social load  $L_s(p_i^k)$  of the partition  $p_i^k$  storing  $u_i$ 's data by using Equation 1. For sake of presentation, we assume  $L_s(p_i^k) = \chi.m$ , where  $\chi$  is the mapping function that expresses the load in terms of response time. Once  $L_s(p_i^k)$  is obtained, we consider two possible states for  $p_i^k$ : acceptable (A) and overloaded (O). The partition is overloaded if  $L_s(p_i^k) \geq T_{max}$ . Given an activity  $a_i$  of  $u_i$ , its neighbors can either partake to it or not, thus we can define a set of Bernoulli random variables  $X = X_1 + X_2 + \dots + X_n$  where  $X_l$  is representing a situation in which actor  $l$  have participated to  $a_i$ . Hence, the sum of such independent variables  $S$  follows the *binomial distribution*  $\sim B(n, p)$ , from which we derive the probability  $p_m$  of having  $m$  actors during an activity.

$$p_m = p(X = m) = \binom{n}{m} p^m (1-p)^{n-m}, \quad (7)$$

where  $p$  is the probability that an actor participates to an activity and  $(1-p)$  the probability it misses it. Moreover, the probability of having  $m$  neighbors interacting during  $N$  activities is

$$\prod_i^N p_m. \quad (8)$$

Once this probability defined, we set a threshold  $\gamma$  beyond which the likelihood of having  $L_s(p_i^k) = \chi.m$  is very high, i.e., the pic load prediction is more accurate. Formally,

$$\prod_i^N p_m \geq \gamma \quad (9)$$

with  $\gamma$  a threshold based on the average participation rate of users. This approach has the advantage to take into account interactions of social network and therefore helps to foresee a peak once some users start interacting.

#### IV. FACING TRANSIENT WORKLOAD

The main idea of facing transient workload is to migrate data of overloaded partitions to a less loaded DB. To this end, we proceed by selective fragmentation and migration that directly takes the overloaded partitions and distribute them to less loaded databases. The problems we face are twofold: 1) identify the database candidate that will receive the extra load and, 2) process the migration mechanism.

##### A. Naive Identification of DB candidates

The basic and naive approach consists of using the less under-loaded DB as candidates to receive extra loads. Basically, when a database is chosen to receive a load from another database, it must remain not overloaded. The naive algorithm of facing a peak of  $DB_k$  works step by step as follows:

- For each  $p_i^k$  in  $M_k$  (see section III-C) that is overloaded, evaluate its load;
- Find all DB candidates that are not overloaded and able to receive  $load(p_i^k)$  without being overloaded afterwards. In fact,  $DB_d$  is a candidate destination to receive  $load(p_i^k)$  if:
 
$$load(DB_d) \leq \frac{T_{max}}{\tau_d} - load(p_i^k) \quad (10)$$
- If there is no database able to receive  $p_i^k$ , then the condition (10) is checked for  $p_{i+1}^k$ .
- After each  $p_i^k$  migration,  $M_k$  is updated. If  $M_k$  still not empty and if no database is able to receive its content then we start a new DB instance and the  $M_k$ 's content is allocated to it.

This approach has the edge to be simple and easy to be implemented and works well for rather regular, slowly fluctuating, workloads. However, in case of stressed workloads generating frequent peaks, this may lead to cascading migrations, i.e., migrating data from a DB that just previously received data from another DB. We need to better anticipate workload fluctuations in order to avoid overloading a DB, which receives an extra load recently. To this end, we propose to take into account the social characteristics of the data. We will exclude a DB candidate that stores data of important users, since it has a high probability to become overloaded in a near future.

##### B. Social-based identification of DB candidates

As mentioned before, important users hold data that are usually the peak origins. An intuitive approach can be to stave off gathering data of several important users in the same database. To this end, we rely on the user interactions graph when migrating data. That is, before moving data of  $DB_1$  to  $DB_2$  we check if the latter does not have important users and if neighbors of such users are likely to participate to activities. In fact, we replace step 2 of the naive approach by an identification method based on the prediction model that uses graph interactions. A DB is a candidate if the data of users it holds respect the model. The other steps are kept unchanged.

##### C. Migration process

We use a similar migration mechanism as in Relational Cloud [5] and ElasTraS [6]. Data is lazily fetched from the source as needed to support transactions on the destination. In-flight transactions are redirected to the destination. Once the data to be migrated and their destination DB are identified, the migration process starts; it consists of two steps:

- First, the initialization step. The source DB informs its associated router and the destination DB. The router spreads this information to the other routers. The indexes, which locate the data partitions are updated. From now, the future transactions on that data are

redirected to the new destination DB. This initiates load balancing. However, at this stage, no data is transmitted yet. Only the indexes are updated and the destination DB is ready for inserting new data when needed.

- Once the initialization step is done, all the transactions accessing the migrated data are routed to the destination DB. When the destination DB receives a transaction on the migrated data, it pulls the pieces of the data that the transaction intends to access, from the source and write it to the DB destination. And so forth, the pieces of the migrated data are transmitted to the destination DB on demand until completion. This pull on demand migration process has the advantages (i) to allow the source DB to alternate data transfer and transaction processing, which reduce the overhead (waiting time) due to the migration; and (ii) to migrate just the needed part of the data. In fact, even if a data have to be migrated, only actually accessed pieces of it are transmitted from the source DB to the destination DB. This aims to reduce the amount of transmitted data, saving communication resource.

- *Preventing continuous migration of a partition* We assume that if a partition  $p_i^k$  of database  $DB_k$  is migrated to another newly initialized  $DB_z$  then the maximum load of that partition  $p_i^k$  could not exceed the  $DB_z$  capacities. That means this partition won't be migrated anymore for overload reasons.

#### D. Example

Given the database  $\{p_1..p_{10000}\}$ . Each  $p_i$  represents one users' data. The data are distributed among three DB nodes  $DB_1$  to  $DB_3$  of various processing capacity. The standalone processing time of a single transaction at  $DB_1$ ,  $DB_2$ , and  $DB_3$  is  $\tau_1 = 20ms$ ,  $\tau_2 = 10ms$ ,  $\tau_3 = 8ms$  respectively. The users require a maximum response time  $T_{max}$  equals to 100ms. Thus, the maximum supported load at  $DB_1$  is  $\frac{T_{max}}{\tau_1} = 5$ . Respectively,  $DB_2$  and  $DB_3$  support a maximum load of 10 and 12.5.

During operation, each DB node process incoming transactions in sequence, queuing pending transactions. We see in Figure 3, two (red colored) transactions pending at node  $DB_1$ : they will wait too long and exceed  $T_{max}$ ; such case requires data migration. To this end, during the last period  $\omega$  (10s), we measure at node  $DB_1$ , the following workload values:  $load(p_1^1) = 1.4$ ,  $load(p_7^1) = 2.1$  and  $load(p_8^1) = 3.5$ , which sums to  $load(DB_1) = 7$ . The amount of extra load at  $DB_1$  is:

$$\Delta_{load_1} = 7 - \frac{T_{max}}{\tau_1} = 2$$

The smaller  $load(p_i^1)$  value greater than  $\Delta_{load_1}$  is  $load(p_7^1) = 2.1$ . Thus, migrating  $p_7^1$  will cause to drop 2.1 of extra load, and to return under  $T_{max}$  response time.

Finally, to find a destination candidate, we measure the amount of  $\Delta_{load}$  that non-overloaded DBs could accept:  $\Delta_{load}$  values are 1 for  $DB_2$ , which is too small compared to our need, and 2.5 for  $DB_3$ , which suits our need. The migration operation give the result we can see in Figure 4.

Note that while estimating the availability a DB node (i.e., the maximum load it may accept), we take into account the

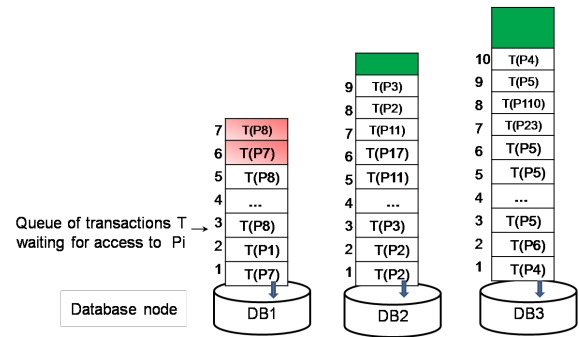


Figure 3: Example: State before migration.

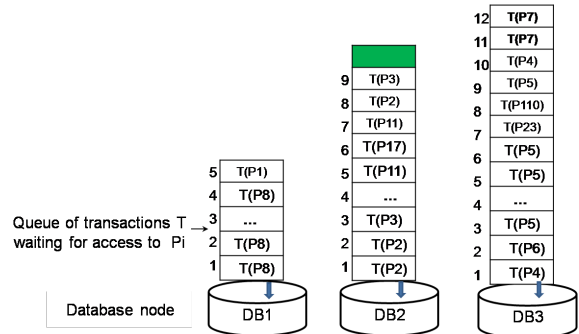


Figure 4: Example: State after migration.

predicted load of its important users so to prevent cascading migrations.

## V. VALIDATION

In this section, we evaluate the overall performances of our approach and we answer the following questions: (i) how long does it take the system to respond to an overload? (ii) What is the cost of facing to an overload? (iii) What is the impact of the reconfiguration on transaction latency? (iv) Does the system can ensure a response time below a given threshold for more than 90 % of cases? (v) How does the number of DB evolve, depending on the workload?

We begin by describing the environment and the tools used during the experiments. Afterwards, we describe the experiments and their results while answering the unveiled questions.

### A. Experimental setup

We use Simjava [7] to simulate our approach and to evaluate it. Simjava is a toolkit (API Java) for modeling complex systems. It is based on discrete events simulation and includes facilities for representing simulation objects. We implement each of entities: *Client nodes (CN)*, *Routeurs*, *Database nodes (DB)* and *the Controller node (CtrlN)*. We use during the simulations, some data structures to represent the storage layer. Each entity is embedded into a thread and exchanges with others through events. During simulation experiments, we focus on the reaction of our system against an heavy and transient workload, and the evolution of the DB nodes. The experiments were conducted on an Intel duo core with 2 GB of RAM and 2.66 GHz running under Ubuntu LTS 12.04.

In the experiments, we implemented the social workload using the algorithm described below. In all experiments, we



started with a small size system (relatively to the number of running machines): one Router and two DB nodes, which store all the data. All transactions are read and write accesses.

### 1) Algorithm for workload generation

We aim to generate the load peaks in a controlled way, ensuring that the load at each partition (i.e.,  $load(p_i^k)$ ) corresponds to a given value. Indeed, the load values at each partition characterizes the workload pattern. We propose an algorithm to generate the workload, which follows the biased (e.g., power-law) pattern of social networks, as described in [1], i.e., the data of users with a high centrality degree receive a higher load. Accordingly, we control how many users are concurrently accessing each partition. Without loss of generality, we assume that a user involved into a peak load is accessing only one partition, as long as the peak occurs. Notice that a user may still access several partitions if her session lasts longer than the peak duration.

More precisely, we design an algorithm to assign users to partitions. We also change this assignment dynamically such that the overall data access frequency follows the same distribution as the underlying centrality degree distribution. Our algorithm takes as input the number  $m$  of partitions, the number  $U$  of users, the peak duration  $D$  in seconds, a distribution function  $f$  (e.g. the zipfian function), and the amount of work to do in a run expressed as the number  $N$  of transactions to process.

- We assess the distribution of the  $N$  transactions to the  $m$  partitions. We use  $f$  to get the number of accesses  $A_i^k$  to each partition  $p_i^k$ , such that the sum of all the  $A_i^k$  values equals  $N$ .
- We distribute the  $U$  users to a subset of the partitions such that there is  $A_i^k$  users per partition, and the sum of the  $A_i^k$  of the partitions in that subset equals  $U$ .
- For  $D$  seconds, every user submits a sequence of transactions to its assigned partition.
- Then we redistribute the users to another subset of the partitions; then we continue the run for  $D$  more seconds, redistribute again, and so on until all the partitions have been accessed by  $A_i^k$  users.

## B. Experiments

### 1) Reaction to unexpected overload

The objective of this experiment is to assess how our system reacts against a higher load and how fast this is done. The results of the experiment are depicted on Figures 5 and 6. Data are horizontally partitioned and each DB node stores 50% of the overall data. We consider a heavy load on a small range of data that does not change during the experiment. Since we use a zipfian distribution to generate this range of hot data, most of them are for the first hundred users and are stored on DB0. We set a threshold of one second for the response time. We observe that the system handles the overload situation a few seconds after it starts: hot data are identified and the load is balanced between DB0 and DB1. Actually, after a peak arrival, one can see it does not take a long time to our system to deal with the peak and to stabilize the latency. The difference between results obtained on DB0 and DB1 is mainly due by

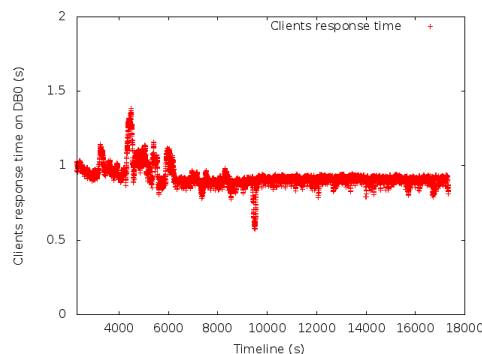


Figure 5: Reaction to a high and subite load at DB0.

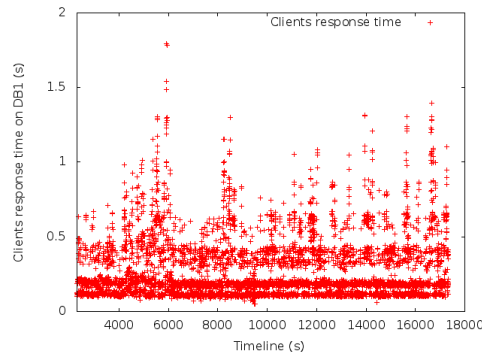


Figure 6: Reaction to a high and subite load at DB1.

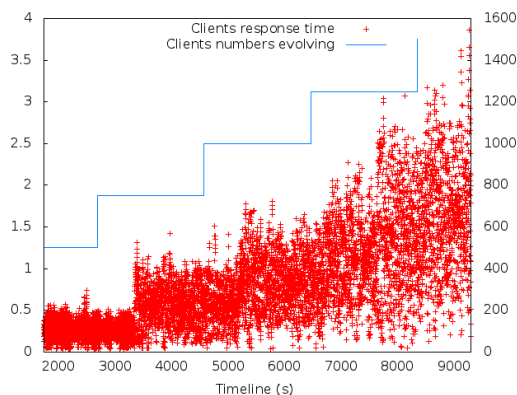


Figure 7: Not controlled: Response times on DB0 increases rapidly while the number of users increases (peaks are on DB0).

the fact that data are moved from DB0 to DB1. That is, it will take more time to DB1 to store arriving data and to process afterwards transactions on such data. Moreover, the workload on DB0 is slightly more dense than the one on DB1.

### 2) Not controlled vs Controlled system

This experiment highlights the impact of controlling the peak load on the performances. In this respect, we compare our system with another one without a peak load management. Results of such comparison are shown on Figures 7 and 8. To this end, the load is gradually increasing from 500 to 1500 users at regular intervals. As expected, the response time increases drastically with a non-controlled system as pointed out by Figure 7. Meanwhile, the response time increases in a logarithmic fashion with our system (see Figure 8).

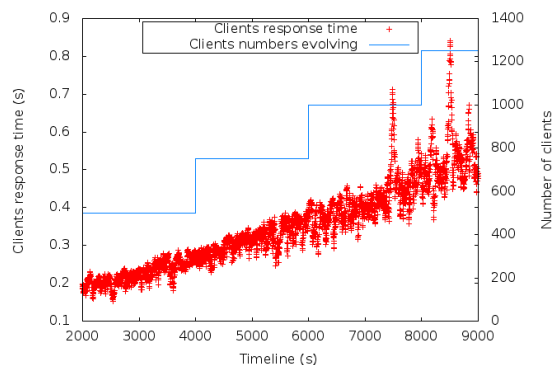


Figure 8: Controlled: Response times on DB0 increases slowly.

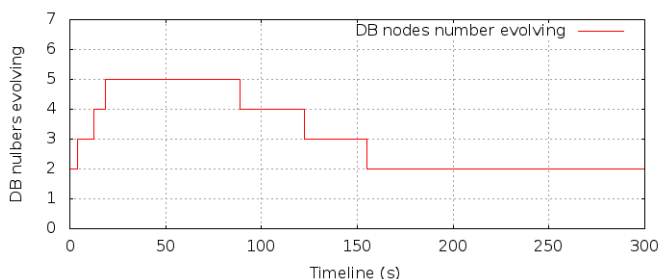


Figure 9: number of DB vs. workload variation.

### 3) Number of resources used

In this experiment, we aim at computing the number of DB used when the load varies. The main goal is to check whether the total number of DB decreases or increases when the workload changes. The load varies from 150 to 300 concurrent users. After a while (i.e., one minute), we reduce the number of clients till 95% of them are off. We set the latency threshold to one second. As one can see on Figure 9, the number of DB nodes grows from 2 to 5 when the workload is increasing. When the workload becomes lighter, the number of DB nodes decreases till we get the initial configuration. Our load balancing algorithm achieves to save computing resources.

#### C. Further experiments and prototype

We still conducting our experiments in a real-world cloud environment: the Amazon EC2. In this thorough experiments, we use the Oracle NoSQL Database for the storage layer and each of entities (i.e., *Client nodes (CN)*, *Routeurs*, *Database nodes (DB)*) is embedded in a virtual machine.

The first obtained results are promising and confirm what we got with SimJava. We will publish this results in an extended subsequent version of this work as soon as possible.

## VI. RELATED WORK

Elasticity and load balancing are essential features to optimize the operation cost in data management systems deployed on a pay-per-use cloud infrastructure, particularly when those data concerns some social media applications. They permit to cope with transient and unpredictable peak loads in the involved system, if they are automated. In this context, several works have been proposed to address the problem of elasticity

[6][8][9][10][11][12]. Most of these studies have adopted the principle of a partitioned database [6][11] and live migration to distribute the load. In particular, Carlo Curino et al. in [8], *ElasTras* [6] and *Albatross* [12] that tackle the problem of minimizing the operating costs of Database systems serving multitenant cloud platforms by efficient resource sharing.

In the literature, most of the work that focused on promoting elasticity in databases are accompanied by migration techniques, and some studies have simply focused on the migration itself. We can mention among them the work done in *Slacker* [9], which uses hot Back-Up tools to copy the database while allowing service continue during this phase. The migration method is based on the available processing capacity on the node source node in order to bound the response time. It is a solution that prevents interruption of the execution during the migration, but it is based on a Back-Up solutions that are database dependent. However, the authors argue that the Back-Up solution is not a problem since their migration is in middleware. Another problem we raise is, as in [10][11][12], the choice of moving a partition (a tenant in this case) without specifying which one. The idea of moving an entire partition is risky if we do not identify which one to move. With the solution such as *Zephyr* [10] and *Albatross* [12], the authors, after identifying the destination, propose a lightweight migration method to move data to their new destination. This migration technique uses an on-demand copy during transaction processing. Thus, they prevent interruptions during transaction processing. The main differences between these solutions and ours are twofold: *i*) we identify the data to move in order to face directly the source of the bottleneck; and *ii*) we migrate only required data. More recently, Jan Schaffner et al. propose RTP: *Robust Tenant Placement for Elastic In-Memory Database Clusters* [13]. This work tackles the minimization of operational costs by proposing algorithms that elastically adapt the system size depending on the tenant behavior. This work differs from ours in the way that they consider a read-intensive workload and use replication on their system.

More generally, there are some data management systems recently produced (less than a decade) [14][15][16][17][18][19][20][21]. These systems are usually oriented to the management of web data (NoSQL, key/value, document, etc..) or transactional data [14]. Some of them provide elasticity mechanisms [15][16][17][18], but their elasticity is rather related to the amount of current data. When it becomes too large, they add a new node and place there a part of the data. This assumes that load is evenly distributed on all the data. This does not fit our context where the load is biased.

Furthermore, elasticity is often coupled with load balancing to minimize resources. In addition, such a minimization of resources has an indirect effect that is the gain in energy consumption. This specific objective called green computing, is reached through the effective use of available resources. In this context, many works was done or are being done to develop new techniques for load balancing [22]. The purpose of load balancing [23][24][25][26][27][28] is an efficient use of resources by redistributing dynamically load through all nodes in the system. Our algorithm is based on such principles for for more effectiveness.

Moreover, some works are conducted for load-balancing while avoiding distributed transactions across multiple partitions [29][30]. These approaches balance data based on current load level on that data. They use graph (hyper-graph in Sword) partitioning algorithm to find a replacement that prevents data distributed transactions. The main objective of these works is to provide an improved throughput while providing fault tolerance and scalability for distributed OLTP data management systems. They do not reflect the economy of resources used as we do.

## VII. CONCLUSION AND FUTURE WORK

We propose to exploit the social structure of online media to face transient heavy workload. Our solution monitors the load level within the database layer and identifies the hot data, which are the sources of peaks load when overload happens. After identifying the origins of peaks load, we proceed by migrating parts of the hot data among the database nodes, with the goal of keeping the transactions response time under a given value. In order to fully make the identification of the sources of peaks load, and choose the right destinations for migration, we have developed fine-grain identification model. Furthermore, we leverage on the social user network to anticipate the load of popular data mostly owned by users with high centrality degree. This allows for early data migration while preventing cascading migration. We validate our approach through experimentation with a synthetic dataset. These experiments show promising performances in terms of resources saved and response time guaranty. Ongoing works are conducted to evaluate our solution on a real-world database workloads for social applications. To this end, we are experimenting on top of Amazon EC2 cloud, using Oracle KVLite [31] for data storage and access at each DB node. We are using the data and the workload from the LinkBench [1] benchmark.

## REFERENCES

- [1] T. G. Armstrong, V. Ponnkanti, D. Borthakur, and M. Callaghan, "Linkbench: a database benchmark based on the facebook social graph," in Intl Conf. on Management of Data (SIGMOD), 2013, pp. 1185–1196.
- [2] M. E. J. Newman, "Fast algorithm for detecting community structure in networks," *Physical Review E*, vol. 69, no. 6, Jun 2004, pp. 1–5.
- [3] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3-5, February 2010, pp. 75–174.
- [4] J. M. McPherson and J. R. Ranger-Moore, "Evolution on a dancing landscape: Organizations and networks in dynamic blau space," *Social Forces*, vol. 70, 1991, pp. 19–42.
- [5] C. Curino, E. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich, "Relational cloud: A database service for the cloud," in Biennial Conf. on Innovative Data Systems Research, 2011, pp. 235–240.
- [6] S. Das, D. Agrawal, and A. El Abbadi, "Elastras: An elastic, scalable, and self-managing transactional database for the cloud," *ACM Trans. Database Syst.*, vol. 38, no. 1, 2013, pp. 5:1–5:45.
- [7] F. Howell and R. Mcnab, "simjava: A discrete event simulation library for java," in Intl Conf. on Web-Based Modeling and Simulation, 1998, pp. 51–56.
- [8] C. Curino, E. P. Jones, S. Madden, and H. Balakrishnan, "Workload-aware database monitoring and consolidation," in Intl Conf. on Management of Data (SIGMOD), 2011, pp. 313–324.
- [9] B. Sean Kenneth, C. Yun, M. Hyun Jin, H. Hakan, and J. S. Prashant, "'cut me some slack': latency-aware live migration for databases," in Intl Conf. on Extending Database Technology (EDBT), 2012, pp. 432–443.
- [10] A. J. Elmore, S. Das, D. Agrawal, and A. E. Abbadi, "Zephyr: Live migration in shared nothing database for elastic cloud platforms," Intl Conf. on Management of Data (SIGMOD), 2011, pp. 301–312.
- [11] M. Umar Farooq, L. Rui, A. Ashraf, S. Kenneth, N. Jonathan, and R. Sean, "Elastic scale-out for partition-based database systems," in IEEE International Conference on Data Engineering Workshops (ICDEW), 2012, pp. 281–288.
- [12] S. Das, S. Nishimura, D. Agrawal, and A. El Abbadi, "Albatross: lightweight elasticity in shared storage databases for the cloud using live data migration," *Proc. VLDB Endow.*, vol. 4, no. 8, 2011, pp. 494–505.
- [13] J. Schaffner, T. Januschowski, M. Kercher, T. Kraska, H. Plattner, M. J. Franklin, and D. Jacobs, "Rtp: robust tenant placement for elastic in-memory database clusters," in Intl Conf. on Management of Data (SIGMOD). ACM, 2013, pp. 773–784.
- [14] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. Zdonik, E. P. C. Jones, S. Madden, M. Stonebraker, Y. Zhang, J. Hugg, and D. J. Abadi, "H-Store: a high-performance, distributed main memory transaction processing system," *Proc. VLDB Endow.*, vol. 1, no. 2, 2008, pp. 1496–1499.
- [15] I. VoltDB, "Voltdb," Retrieved on March 2014. [Online]. Available: <http://voltdb.com>
- [16] I. Basho Technologies, "Riak," Retrieved on March 2014. [Online]. Available: <http://docs.basho.com>
- [17] F. Apache Software, "Apache hbase," Retrieved on March 2014. [Online]. Available: <http://hbase.apache.org/>
- [18] I. CouchBase, "Apache couchdb," Retrieved on March 2014. [Online]. Available: <http://www.couchbase.com/>
- [19] F. Apache Software, "Apache couchdb," Retrieved on March 2014. [Online]. Available: <http://couchdb.apache.org/>
- [20] m. Inc., "mongodb," Retrieved on March 2014. [Online]. Available: <http://www.mongodb.org/>
- [21] I. Amazon Web Services, "Amazon dynamodb," Retrieved on March 2014. [Online]. Available: <http://aws.amazon.com/fr/dynamodb/>
- [22] I. C. Nidhi Jain Kansal, "Cloud load balancing techniques: A step towards green computing," *International Journal of Computer Science Issues (IJCSI)*, vol. Vol. 9, Issue 1, No 1, 2012, pp. 238–246.
- [23] A. M. Nakai, E. Madeira, and L. E. Buzato, "Load balancing for internet distributed services using limited redirection rates," *IEEE Latin-American Symposium on Dependable Computing (LADC)*, 2011, pp. 156–165.
- [24] Y. Lua, Q. Xiea, G. Kliotb, A. Gellerb, J. R. Larusb, and A. Greenber, "Join-idle-queue: A novel load balancing algorithm for dynamically scalable web services," *Intl Journal on Performance evaluation*, 2011, pp. 1056–1071.
- [25] H. Mehta, P. Kanungo, and M. Chandwani, "Decentralized content aware load balancing algorithm for distributed computing environments," in Intl Conf. Workshop on Emerging Trends in Technology (ICWET), 2011, pp. 370–375.
- [26] B. Jasma and R. Nedunchezian, "A hybrid policy for fault tolerant load balancing in grid computing environments," *Journal of Network and Computer Applications*, 2012, pp. 412 – 422.
- [27] G. You, S. Hwang, and N. Jain, "Scalable load balancing in cluster storage systems," in *Middleware*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2011, vol. 7049, pp. 101–122.
- [28] H. T. Vo, C. Chen, and B. C. Ooi, "Towards elastic transactional cloud storage with range query support," *VLDB Endow.*, 2010, pp. 506–514.
- [29] C. Curino, E. Jones, Y. Zhang, and S. Madden, "Schism: a workload-driven approach to database replication and partitioning," *VLDB Endow.*, vol. 3, no. 1-2, 2010, pp. 48–57.
- [30] Q. Abdul, K. Kumar, and A. Deshpande, "Sword: Scalable workload-aware data placement for transactional workloads," in Intl Conf. on Extending Database Technology (EDBT), 2013, pp. 430–441.
- [31] C. Oracle, "Oracle nosql database," Retrieved on November 2013. [Online]. Available: <http://www.oracle.com/technetwork/products/nosqldb/overview/index.html>

## Sample Trace: Deriving Fast Approximation for Repetitive Queries

Feng Yu

Department of Computer Science  
and Information System  
Youngstown State University  
Email: fyu@ysu.edu

Wen-Chi Hou

Department of Computer Science  
Southern Illinois University Carbondale  
Email: hou@cs.siu.edu

Cheng Luo

Department of Mathematics and  
Computer Science  
Coppin State University  
Email: cluo@coppin.edu

**Abstract**—Repetitive queries refer to those queries that are likely to be executed repeatedly in the future. Queries such as those used to generate periodic reports, perform routine summarization and data analysis belong to this category. Repetitive queries can constitute a large portion of the daily activities of a database system, and thus deserve extra optimization efforts. In this paper, we propose to record information about how tuples are joined in a repetitive query, called the *query trace*. We prove that the query trace is sufficient to compute the exact selectivities of joins for all plans of a given query. To reduce the space and time overheads in generating the query trace, we propose to construct only a sample of the query trace, called a *sample trace*, which can be much smaller than a (complete) query trace. A special operation, called a *sample outer join*, is designed to accomplish this feat. Accurate estimations of join selectivities, with associated confidence intervals, can be derived easily using the sample trace. Extensive experiments show that the sample trace can be constructed efficiently and be a controllable trade-off between accuracy and efficiency in estimations of join selectivities for repetitive queries.

**Keywords**—query optimization, query re-optimization, trace, sampling method, sample trace

### I. INTRODUCTION

Query re-optimization aims to refine execution plans of queries. There has been some progress made on this subject recently. In the literature, some [1–3], have focused on refining execution plans on-the-fly for currently running queries, while others [4–8], etc., on refining cost estimation for future queries. In this paper, we are interested in identifying useful statistics for refining cost estimation for future queries, similar to the latter.

To refine cost estimation for future queries, a common approach is to collect actual selectivities [7], [8], and some other statistics of the operators [5] and use them to adjust cost estimation for future queries. Unfortunately, selectivities of joins obtained from one plan of a query may not be sufficient for estimating selectivities of joins of another plan of the same query because as the join order changes, the selectivities of joins change, not to mention the selectivities of joins of other queries. The problem is exacerbated by adding selection predicates to the queries (to specify the tuples of interest). It is probably difficult to gather all information that may be needed for estimating selectivities of joins for all plans of all possible queries. Therefore, in this research, we set a more realistic goal by restricting ourselves to considering only a subset, but a large and frequently used subset, of queries, called repetitive

queries.

Repetitive queries refer to those that are likely to be posted repeatedly in the future. Many useful queries, such as those used for generating periodical reports, performing routine maintenances, summarizing and grouping data for analysis, are repetitive queries. They are often stored in databases for convenient reuses for the long term. Any sub-optimality in the execution plans of such queries may mean repetitive and continued waste of system resources and time. Moreover, as new queries are continuously being developed, more queries become repetitive queries. Usually, the longer a database is in production, the greater the number of repetitive queries is in the database; their executions can constitute a large part of daily activities of a database system. The optimality of execution plans of repetitive queries has a tremendous effect on the performance of the system and thus deserves more optimization efforts.

Unlike much of the existing re-optimization work that focuses on refining physical execution plans of queries, our research focuses on refining logical execution plans (or the join order) of queries. Selectivities obtained from previous executions can be very useful for selecting an efficient access method (e.g., table scan and index access) and join method (e.g., nested-loop, sort-merge, etc.), that is, physical plans, but they may not be sufficient to estimate the selectivities of joins of the query in other join orders accurately without using simplifying assumptions, such as attribute independence and/or distribution uniformity. Our approach here focuses on how to gather sufficient information for computing the selectivities of joins for all plans of a query accurately.

In this paper, we continue to study the query or join trace [9], [10] that records information about how tuples are joined in the query. We have designed operators to gather such information so that the exact join selectivities in all join orders for a query can be computed. To reduce the overheads incurred in collecting a query trace, we design an innovative sample scheme that generates only a sample of the query trace. The sampling scheme is very effective in reducing overheads and the experimental results have shown that the sample trace is very accurate. With accurate selectivity estimations, running repetitive queries in the most efficient ways becomes possible.

The rest of the paper is organized as follows. Section II discusses previous work in relation to re-optimization. Section III introduces the query trace. The relationships between the traces and selectivities of queries with acyclic join graphs are

discussed in Sections IV. In Section V, we discuss deriving a sample of the trace and using the sample trace to estimate the selectivity of an arbitrary subquery. The sample trace is empirically evaluated in Section VI. Section VII presents the conclusions and future work.

## II. LITERATURE SURVEY

The work in query re-optimization can be classified into two categories: (1) re-optimizations of current (or ongoing) queries and (2) re-optimization of future queries. Our work falls into the second category as we try to optimize the future executions of repetitive queries.

There has been much work that falls into the first category. ReOpt [3] discussed re-optimization of join queries. Statistics are collected at run-time and ad hoc heuristics are used to determine whether or not to re-optimize by either changing the execution plan or improving the resource allocations for the remainder of the execution. POP [11] improved upon ReOpt [3] by computing a more rigorous validity range for an input to a join within which the plan is valid. When the actual cardinality falls outside the validity range, a re-optimization is triggered. Rio [1] further improved on POP by computing interval estimates, instead of point estimates, of the cardinalities. Within the intervals, they selected robust and switchable plans to avoid repeated re-optimization and loss of earlier pipelined work.

The second category includes [4], [11–15] Chen et al. [14] first used query feedback to refine the data distribution, represented by a linear combination of “model function”. Only 1-dimensional distributions were considered. Abounaga et al. [4] used the actual range selectivities obtained from queries to adjust histograms. By splitting high frequency buckets and merging similar consecutive buckets, histograms are tuned. Lim et al. [6] used actual selectivities to tune bucket frequencies and query workloads to determine what set of histograms to build. All these approaches are mainly concerned with selection queries.

The pay-as-you-go approach [16] improves the idea of LEO [7], [8] with proactive monitoring and plan modifications. Nevertheless, it does not collect enough information needed for estimating arbitrary logical execution plans, and it must rely on repetitive executions of the query to collect complementary cardinality information.

Microsoft Index Wizard [17] recommends the database administrator (DBA) on indexes. It is similar to DB2 Advisor [18] and Oracle SQL Access Advisor [19] that are limited to access path recommendations rather than selections of logical execution plans.

Xplus [20] enumerates plans and their neighborhoods to search for alternative plans with lower cost. However, it may only find plans with local minimum values that may not necessarily be the optimal plan.

Oracle’s Automatic Tuning Optimizer (ATO) [21] performs SQL Profiling and what-if analysis on selected high load SQL statements. SQL Profiles are used with other statistics to build a well-tuned plan. However, it may not have sufficient statistics that are needed for cost estimation of all plans of the query, and must generate auxiliary information for estimations, which can be time-consuming and inaccurate.

In this research, we are interested in gathering sufficient information about a query during execution so that an optimizer can use the information to find the best join order, or the best

logical execution plan, for the query. Existing re-optimization works, such as POP, Rio, and LEO can be very effective in refining physical execution plans of queries, e.g., access methods (e.g., table scan, and index access and join methods (e.g., nested loop, sort-merge, and hash join using the actual selectivities obtained from the feedback. However, they may fall short of optimizing logical execution plans of the queries because as the join orders change, the selectivities of joins in alternative plans change too. It requires more information than just the selectivities of operators of the current plan to estimate the selectivities of joins of alternative plans accurately.

## III. QUERY TRACE

When a query is being processed, information about how tuples are joined is gathered. We intend to use this information, called the query or join trace, to estimate selectivities of joins in all execution orders.

We use tuple IDs to identify matching tuples in the joins of a query in the trace. To create the trace of a query, an ID attribute,  $R_i$ -ID, is added to (the schema of) each relation  $R_i$ . The added ID attributes are to be included in the output (schema) of all operations to identify tuples that contribute to the output. For example, in the join of two relations  $R_1$  and  $R_2$ , a result tuple, besides its normal set of attributes, will have two additional attributes:  $R_1$ -ID and  $R_2$ -ID, whose values identify the pairs of tuples that match in the join of  $R_1$  and  $R_2$ . It is noted that we do not need to add an ID attribute physically to (the schema of) a relation on disk, but just append an ID value when a tuple flows into the join operation.

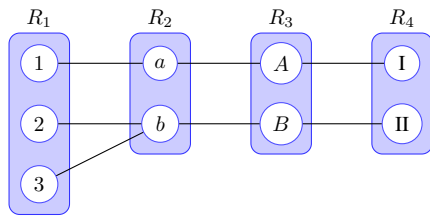
In the following, we use an example to show how query traces look like in their simplest form and how they are generated. More complicated examples will be discussed in subsequent sections, including using other operators, such as outer joins, or designing new operators, such as extended outer joins, to gather sufficient information in the trace for different types of queries.

**Example 1. (Trace)** Consider a left-deep tree execution plan  $P = ((R_1 \bowtie R_2) \bowtie R_3) \bowtie R_4$ . To generate the trace, an ID attribute is added to every relation and the attribute is to be preserved in the outputs of all operators. Thus, the result of  $R_1 \bowtie R_2$ , as shown in Fig. 1(b), besides its normal set of attributes, denoted by Result-Attrs, has additional attributes  $R_1$ -ID and  $R_2$ -ID, called the *trace* of  $R_1 \bowtie R_2$ , denoted by  $T(R_1 \bowtie R_2)$ .

Once a query is completely processed, we can extract the final trace, e.g.,  $T(((R_1 \bowtie R_2) \bowtie R_3) \bowtie R_4)$  in Example 1. Keeping the final trace is enough to derive the selectivities of joins in all execution orders. Consequently, we shall retain only the final trace for selectivity re-estimation. *Hereafter, the trace of a query refers to the final trace of the query, unless otherwise stated.*

## IV. SELECTIVITY ESTIMATION FOR ACYCLIC JOIN GRAPHS USING TRACE

Let  $Q$  be a query with an acyclic join graph  $G(V, E)$  and  $P$  an execution plan of the query. Let  $T(P)$  be the final trace of  $P$ . Let  $G'(V', E')$  be a vertex-induced connected subgraph of  $G(V, E)$ , where  $V' = \{R_{i_1}, \dots, R_{i_m}\} \subseteq V$  and  $E' \subseteq E$ , representing a subquery  $Q'$  of  $Q$ . We propose to compute the exact selectivity of  $Q'$  as



(a) Matching of Tuples

Result-Attrs	$R_1$ -ID	$R_2$ -ID
...	1	a
...	2	b
...	3	b

 (b) Result and Trace of  $R_1 \bowtie R_2$ 

$R_1$ -ID	$R_2$ -ID	$R_3$ -ID
1	a	A
2	b	B
3	b	B

 (c) Trace of  $(R_1 \bowtie R_2) \bowtie R_3$ 

$R_1$ -ID	$R_2$ -ID	$R_3$ -ID	$R_4$ -ID
1	a	A	I
2	b	B	II
3	b	B	II

 (d) Trace of  $((R_1 \bowtie R_2) \bowtie R_3) \bowtie R_4$ 

Figure 1. Query Traces with Tuple IDs

$$\widetilde{sel}(Q') = \frac{|\pi_{R_{i_1}\text{-ID}, \dots, R_{i_m}\text{-ID}} T(P)|}{|R_{i_1}| \times \dots \times |R_{i_m}|} \quad (1)$$

where  $\pi_{R_{i_1}\text{-ID}, \dots, R_{i_m}\text{-ID}} T(P)$  is the projection of trace  $T(P)$  on attributes  $R_{i_1}\text{-ID}, \dots, R_{i_m}\text{-ID}$ , with duplicates removed.

As we shall discuss later that trace tuples, generated by other operators (e.g., (extended) outer joins), could have null values in some of the projected ID components  $R_{i_1}\text{-ID}, \dots, R_{i_m}\text{-ID}$ , and such tuples shall not be accounted for in  $|\pi_{R_{i_1}\text{-ID}, \dots, R_{i_m}\text{-ID}} T(P)|$ .

The cardinalities of the input relations  $|R_{i_1}|, \dots, |R_{i_m}|$  can be obtained easily. If  $R_{i_j}, 1 \leq j \leq m$ , is a base relation,  $|R_{i_j}|$  is certainly known. If  $R_{i_j}$  represents a relation that is immediately preceded by some selections and projection,  $|R_{i_j}|$  can be obtained by counting the number of tuples flowing into the join operation. Another possible way to compute  $|R_{i_j}|$  is just to count the numbers of unique IDs in the respective ID columns in the final trace. Thus, without regard to the access methods used to retrieve tuples from relations, such as table scan, index access, etc., exact  $|R_{i_j}|$  can always be obtained.

#### A. No Dangling Tuples in the Joins

Here, we assume no dangling tuple exists in any of the joins in the query. The join relationships in Fig. 1(a) satisfy this condition. Now, let us see how accurate (1) derives the selectivities.

**Example 2.** (No Dangling Tuple). Consider the query and plan in Example 1. Given the (final) query trace in Fig. 1(c), the selectivities of  $R_1 \bowtie R_2$ ,  $R_2 \bowtie R_3$ , and  $R_3 \bowtie R_4$ , are derived, by (1), as  $\frac{1}{2}$ ,  $\frac{1}{2}$ , and  $\frac{1}{2}$ , respectively, which are the exact selectivities of the respective joins. The derived selectivities of  $(R_1 \bowtie R_2) \bowtie R_3 (= (R_2 \bowtie R_3) \bowtie R_1 = (R_3 \bowtie R_2) \bowtie R_1)$ ,  $(R_3 \bowtie R_4) \bowtie R_2 (= (R_3 \bowtie R_2) \bowtie R_4 = (R_4 \bowtie R_3) \bowtie R_2)$ , are all  $1/4$ ; again they are all exact.

It is not a coincidence that the estimated selectivities are exact. In fact, we can prove that if there is no dangling tuple in any join of the query, (1) derives the exact selectivities of all possible subqueries.

#### Theorem 1 (Exact Estimation without Dangling Tuples).

Let  $P$  be an execution plan of a query  $Q$  with a connected acyclic join graph  $G(V, E)$ . Let  $Q'$  be a subquery of  $Q$  that has a vertex-induced connected join subgraph  $G'(V', E')$ ,  $V' = \{R_{i_1}, \dots, R_{i_m}\} \subseteq V$ . If there is no dangling tuple in any join of  $P$ , (1) gives the exact selectivity of  $Q'$  from  $T(P)$ .

*Proof:* Please see Appendix A. ■

#### B. Dangling Tuples in Joins

Now, let us consider joins with dangling tuples. Dangling tuples are lost in the joins. To retain matching information about dangling tuples, we replace the joins in the original query by the full outer joins ( $\overset{\circ}{\bowtie}$ ). Fig.2(c) to 2(e) show the traces generated at different stages of query execution, where the joins are replaced by the full outer joins. The trace in Fig. 2(c) is the same as if it were generated by a join because there is no dangling tuple in the join. The trace in Fig. 2(d) retains information about dangling tuples b in  $R_2$  and B in  $R_3$  by the outer join.

The estimated selectivities for  $R_1 \bowtie R_2$ ,  $R_2 \bowtie R_3$ , and  $R_3 \overset{\circ}{\bowtie} R_4$  are now, by (1),  $\frac{1}{2}$  ( $= \frac{2}{2 \times 2}$ ),  $\frac{1}{4}$  ( $= \frac{1}{2 \times 2}$ ), and  $\frac{1}{2}$  ( $= \frac{2}{2 \times 2}$ ), respectively, which are exact. Note that, as mentioned earlier, a trace tuple having a null for any of the projected attributes is not accounted for in the respective  $|\pi_{R_{i_1}\text{-ID}, \dots, R_{i_m}\text{-ID}} T(P)|$  because a null in a  $R_{i_j}\text{-ID}$  column of a trace tuple indicates that there is no match found in  $R_{i_j}$  for the respective combination of tuples to generate an output in the (sub)query. One can easily verify that the estimated selectivities for all other subqueries are all exact.

#### Theorem 2 (Exact Estimation with Dangling Tuples).

Let  $P$  be an execution plan of a query  $Q$  with a connected acyclic join graph  $G(V, E)$ . Let  $Q'$  be a subquery of  $Q$  that has a vertex-induced connected join subgraph  $G'(V', E')$ ,  $V' = \{R_{i_1}, \dots, R_{i_m}\} \subseteq V$ . (1) gives the exact selectivity of  $Q'$  from the trace obtained by replacing the joins in the query with the full outer joins, denoted by  $T(P)$ .

*Proof:* Please see Appendix B. ■

For simplicity, hereafter an outer join refers to a full outer join, unless otherwise stated. Note also that we have used  $T(P)$  to denote the trace of a query, regardless of whether the trace is generated by the joins, outer joins, or even other operators (to be discussed shortly).

## V. SAMPLE TRACE AND SELECTIVITY ESTIMATION

Dangling tuples are retained in the outer and extended outer joins. Thus, the result trace can contain more tuples than the query result. This situation is exacerbated when relations are preceded by selection predicates, which can eliminate matching tuples from operand relations. In this section, we discuss a sampling design that not only can significantly reduce the size of result trace, but it also can provide accurate selectivity estimations.

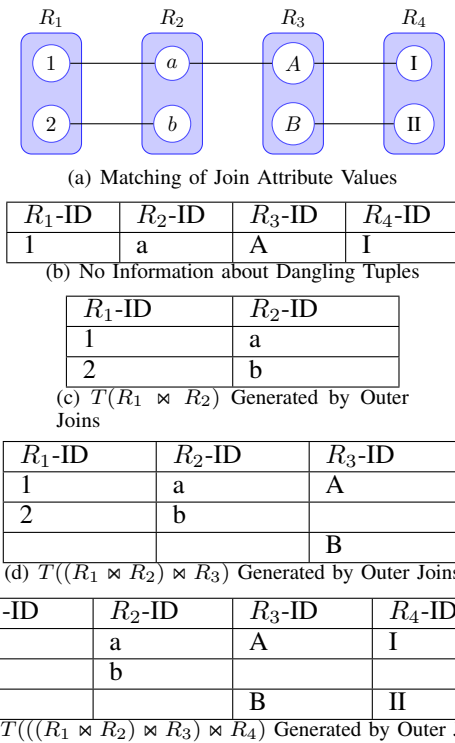


Figure 2. Dangling Tuples in Relations

### A. Sample Trace

A set of uniform random sample tuples is designated for each relation. Instead of retaining all dangling tuples in the trace, we keep only those dangling tuples that are related to sample tuples. Certainly, query result tuples must be kept as before. This partial trace is called a **Sample Trace**.

**Example 3.** (Sample Trace) Consider Fig. 3(a). Assume tuple 1,  $a$ ,  $B$ , and I (in green and with asterisk) are picked as sample tuples respectively from  $R_1$ ,  $R_2$ ,  $R_3$ , and  $R_4$ . We intend to generate trace tuples only for query result tuples and dangling tuples that are related to the sample tuples. In other words, only those trace tuples containing any of 1,  $a$ ,  $B$ , I, or result tuples will be generated. That is, only the first and third rows of the original trace (red lines in Fig. 3(a)) will be generated to be the *Sample Trace*, as shown in Fig. 3(b).

**Definition 1. (Sample Trace)** Given a join query  $Q$  with the plan  $((R_1 \bowtie R_2) \dots \bowtie R_i) \dots \bowtie R_n$ , and  $\{S_i\}_{i=1}^n$  are the simple random samples of  $\{R_i\}_{i=1}^n$ . The **Sample Trace** of  $Q$ ,  $T^*$ , is the subset of the trace tuples of  $Q$  that are generated by sample tuples from  $\{S_i\}_{i=1}^n$ .

To generate such a sample trace, the outer join operator must be modified. The modified operator is called a **Sample Full Outer Join** or just a **Sample Outer Join**. If the input tuple is a sampled tuple or its joinable with a sample tuple, then the sample outer join performs same as an outer join; otherwise, it performs like the ordinary join operation. That is, a dangling tuple in a join will become a result tuple of the sample outer join only if it is a sample tuple from a base relation or it is an intermediate result tuple derived from a sample tuple. Certainly, a pair of match tuples generates an output tuple in the sample outer join, like in an ordinary join.

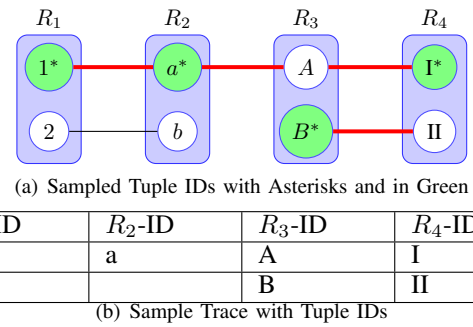


Figure 3. An Example of Sample Trace

```

1: for each tuple  $t_r \in R_r$  do
2:   for each match  $t_l \in R_l$  do
3:     output a result tuple  $t$ 
4:     tag[ $t$ ] = tag[ $t_l$ ] OR tag[ $t_r$ ]
5:     if no match found in  $R_l$  and tag[ $t_r$ ] = 1 then
6:       output a result tuples with nulls for all attributes
         of  $R_r$ 
7:     end if
8:   end for
9: end for
10: for each tuple  $t_l \in R_l$  that found no match in  $R_r$  and
    tag[ $t_l$ ] = 1 do
11:   Output a result tuple with null for all attributes of  $R_l$ .
12: end for
    
```

Figure 4. Algorithm of Sample Outer-join

Figure 4 describes the procedure of Sample Outer-join ( $R_l$ ,  $R_r$ ). Let  $R_l$  and  $R_r$  be the left and right operand relations of a sample outer join. A boolean tag tag[ $t$ ] is associated with each tuple  $t$ , with tag[ $t$ ] = 1 indicating that  $t$  is a sample tuple or is derived from a sample tuple; tag[ $t$ ] = 0, otherwise.

### B. Selectivity Estimation

Consider a query  $Q$  with the plan  $((R_1 \bowtie R_2) \dots \bowtie R_i) \dots \bowtie R_n$ . Let  $T^*$  be its sample trace generated by replacing the joins with the sample outer joins. Given a subquery  $q$  involving  $R_{i_1}, \dots, R_{i_j}, \dots, R_{i_k}$ ,  $1 \leq j \leq k$ , we attempt to estimate the result size of  $q$  using  $T^*$ . More precisely, we will use only a subset of  $T^*$  that is related to the sample tuples from  $R_{i_1}$ , to estimate the query size.

Let  $S_{i_1}$  be the set of IDs of sample tuples from  $R_{i_1}$ . We denote the subset of  $T^*$  that is related to the sample tuples from  $R_{i_1}$  by  $T_{i_1}^*$ , that is,  $T_{i_1}^* = \sigma_{R_{i_1}\text{-ID} \in S_{i_1}}(T^*)$ . Let  $n_{i_1}$  be the number of result tuples of  $q$  generated by the sample tuples from  $R_{i_1}$ . It can be observed that

$$n_{i_1} = \left| \prod_{R_{i_1}\text{-ID}, \dots, R_{i_k}\text{-ID}} T_{i_1}^* \right| \quad (2)$$

where  $\prod$  is a projection operation. As mentioned, a tuple with a null in any of the attributes  $R_{i_1}\text{-ID}, \dots, R_{i_k}\text{-ID}$  will be removed in the above projection. Our estimation formula for the subquery  $q$  is stated in the following theorem.

**Theorem 3 (Unbiased Estimation with Sample Trace).** Consider a query  $Q$  with the plan  $((R_1 \bowtie R_2) \dots \bowtie R_i) \dots \bowtie R_n$ , and a subquery  $q$  involving  $R_{i_1}, \dots, R_{i_j}, \dots, R_{i_k}$ ,  $1 \leq j \leq k$ .

The query result size of  $q$ ,  $Y_q$ , is estimated as

$$\hat{Y}_q = n_{i_1} \frac{|R_{i_1}|}{|S_{i_1}|} \quad (3)$$

which is an unbiased estimator of the query result size of subquery  $q$ . Here,  $|R_{i_1}|$  and  $|S_{i_1}|$  are the sizes of  $R_{i_1}$  and  $S_{i_1}$ , respectively. And  $n_{i_1}$  is described in (2).

*Proof:* Please see Appendix C. ■

By Theorem 2.2 in [22], we get the variance of the query estimation using sample trace in the following theorem.

**Theorem 4 (Variance of Estimator).** *Let*

$$S^2 = \frac{\sum_{j=1}^{|R_{i_1}|} (Y_j - \bar{Y}_q)^2}{|R_{i_1}| - 1}$$

in which  $Y_j$  is the total number of result tuples generated by the  $j$ th tuple in relation  $R_{i_1}$  in subquery  $q$ , and  $\bar{Y}_q = \frac{1}{|R_{i_1}|} \sum_{j=1}^{|R_{i_1}|} Y_j$ . The variance of the query size estimation for subquery  $q$  using sample trace is

$$\text{Var}(\hat{Y}_q) = \frac{|R_{i_1}|^2}{|S_{i_1}|} S^2 \left(1 - \frac{|S_{i_1}|}{|R_{i_1}|}\right)$$

We can also get an unbiased estimation of  $\text{Var}(\hat{Y}_q)$  from the values in the sample relations. By Theorem 2.4 in [22], we have the following theorem.

**Theorem 5 (Approximation of Variance).** *Let*

$$s^2 = \frac{\sum_{j=1}^{|S_{i_1}|} (y_j - \bar{y})^2}{|S_{i_1}| - 1}$$

in which  $y_j$  is the total result tuples generated by the  $j$ th tuple in the sample relation  $S_{i_1}$ , and  $\bar{y} = \frac{n_{i_1}}{|S_{i_1}|}$ . An unbiased estimation of  $\text{Var}(\hat{Y}_j)$  is

$$v(\hat{Y}_q) = \frac{|R_{i_1}|^2}{|S_{i_1}|} s^2 \left(1 - \frac{|S_{i_1}|}{|R_{i_1}|}\right)$$

It is usually assumed that the estimated value  $\hat{Y}_q$  is normally distributed about the corresponding true query result size  $Y_q$ . When this assumption holds, by Central Limit Theory [22], we derive the confidence interval of  $Y_q$  as follows.

**Theorem 6 (Confidence Interval of  $Y_q$ ).** *The associated confidence interval of  $Y_q$  can be computed as*

$$\left(\hat{Y}_q - t_p \sqrt{v(\hat{Y}_q)}, \hat{Y}_q + t_p \sqrt{v(\hat{Y}_q)}\right)$$

where  $v(\hat{Y}_q)$  is in Theorem 5 and  $t_p$  is the normal deviate corresponding to the desired probability  $p$ .

## VI. EXPERIMENTAL RESULTS

The experiments will focus on the feasibility of using sample traces by examining their estimation accuracy and construction overheads, including space and time. We replace the joins in a query by sample outerjoins to construct the sample trace and use it to estimate the selectivities of joins in all possible orders for the query. All programs are implemented in Java on a X86 Linux Desktop, equipped with a 3.4 GHz CPU, a 4GB RAM, and a 500GB hard drive (7200rpm, buffer size 16MB). All data, including datasets, test queries, and

intermediate query results, is materialized on a local hard drive. To facilitate the evaluation of sample outerjoins, we use index files.

### A. Datasets and Sampling Ratios

We conducted experiments on both synthetic data — the TPC-H skewed datasets [23], and real-life data — the DBLP dataset [24]. We generated 1GB TPC-H datasets with different skew factors of  $z = 0.5, 1, \text{ and } 1.5$ . Each TPC-H dataset has 8 relations. The DBLP database listed more than 1.3 million articles in computer science. The DBLP dataset sources from a relational enhancement of the original DBLP data, named DBLP++ [25]. The many-many relationship between “author” and “paper” has been replaced by introducing an “author\_of” relation and two many-one relationships: from “author\_of” to “author”, and “author\_of” to “paper”, as it is often did in relational databases.

Sampling ratio indicates the fraction of tuples that are sampled to construct the sample trace. We performed experiments with sampling ratios from 0.1%, 0.2%, ..., to 1% to test the average performance, as shown in Fig. 5.

### B. Test Queries

A set of test queries is constructed for each dataset. For the TPC-H datasets, we chose a subset of the original TPC-H benchmark queries that contain joins of 3 to 8 selections. The total number of queries is 15 from which we construct 50 subqueries, with different join orders, to examine the accuracy and overheads. For the DBLP dataset, we generated 5 queries that involved joins of all relations in the database. 20 subqueries are tested. Note that, selection predicates are deployed on relations. The ranges of selection predicates are randomly generated to examine the average performance of the sample traces under complex selection conditions.

### C. Space Performance

Besides the query result tuples, the sample outer join retains dangling tuples that are related to the sample tuples in evaluation, which contributes to the space overhead.

The space overhead, denoted  $SO$ , of the sample trace is calculated as

$$SO = \frac{\#\{\text{intermediate dangling tuples}\}}{\#\{\text{intermediate query result sizes}\}} \times 100\%$$

in which “ $\#\{\}$ ” denotes the cardinality of a set, and “intermediate” means that we account for all sample traces generated during the intermediate phases of a query.

The space overheads for datasets TPCH1G05 and TPCH1G1 are very close. It is because their skew factors are relatively low ( $z = 0.5$  and 1). When the skew factor  $z$  increased to 1.5 (i.e., very skewed) in TPCH1G15, the space overhead in Fig. 5(a) grew noticeably higher than the other two low skewed TPC-H datasets ( $z = 0.5$  and 1). Under the same selection ranges in the test queries, greater skew factor produced more dangling tuples, as explained earlier.

Results on the DBLP dataset are similar. The space overhead increased with the increase of the sampling ratio. The space overhead was affected by the skewness of the dataset and the selection criteria of the queries.

### D. Accuracy Performance

The results of the accuracy tests are shown in Fig. 5(b). We use the absolute relative error, or relative error,  $E$ , to evaluate



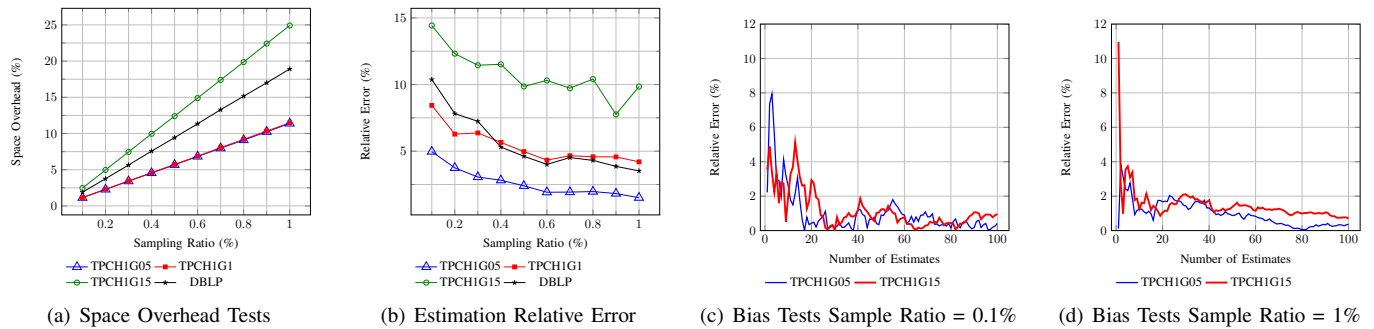


Figure 5. Sample Trace Tests on Multiple Datasets

TABLE I. AVERAGE PERFORMANCE

	TPCHIG05	TPCHIG1	TPCHIG15	DBLP
Space Overhead (%)	6.26	6.32	13.68	10.40
Construction Time (s)	0.06	0.06	0.05	0.30
Time Overhead (%)	1.10	1.12	1.06	5.90
Relative Error (%)	2.62	5.41	10.76	5.56

the accuracy of using a sample traces, which is defined as follows.

$$E = \left| \frac{\widehat{Y}^{Est} - Y^{Act}}{Y^{Act}} \right| \times 100\%$$

where  $Y^{Act}$  denotes the actual query result size and  $\widehat{Y}^{Est}$  is the estimated result size.

To evaluate the average performance, for each test query, we generated 10 sets of sample traces, and use them to estimate the sizes of the subqueries of these test queries.

As shown in Fig. 5(b), the relative errors of all datasets decreased gradually as the sampling ratio increased. The relative errors for datasets TPCHIG05 ( $z = 0.5$ ), TPCHIG1 ( $z = 1.0$ ), and DBLP decreased more smoothly than the relative errors for TPCHIG15 ( $z = 1.5$ ), which has a higher skew factor than others. In general, it is more difficult to represent a skewed distribution than a smooth distribution. Therefore, the more skewed the data, the higher the sampling ratio is needed to achieve the same accuracy.

#### E. Average Performance

The average performances of sample trace on all datasets are listed in Table I. Note that the minimum sampling ratio is 0.1% and the maximum is 1%. The space overheads are less than 13.68%, and construction times are no more than 0.3 seconds. In addition, under a low sampling ratio, the maximum relative error is 10.76%, which occurred on TPCHIG15 dataset when the sampling ratio is 0.1%.

In summary, with a small amount of samples (e.g.,  $\leq 1\%$ ), accurate estimation of subquery result sizes (e.g.,  $\leq 10\%$ ) can be obtained, even for highly skewed data. And the time and space overheads are mostly small, around 1% and 10%, respectively. The experimental results confirm that sample trace can be a viable approach for re-estimation of selectivities for repetitive queries. It is effective and accurate.

#### F. Bias Test of the Sample Trace Estimator

In Section V, we proposed using  $\widehat{Y}_q$  to estimate a subquery with sample trace, which is proved to be unbiased in Theorem 3. Here we empirically validate the unbiasedness of  $\widehat{Y}_q$ . For demonstrative purpose, we choose one query and one of its subqueries from previous test queries. We performed the

unbiased tests on two datasets TPCHIG05 and TPCHIG15, where the sample ratios are equal to 0.1% and 1%, respectively. A subquery is estimated using a different sample trace of the same sample ratio. We generated 100 sample traces for this purpose. To demonstrate the variation tendency of bias when the number of estimations increases, we average the first  $i$  estimation values by absolute average relative error defined as

$$E_i = \left| \frac{\frac{\sum_{j=1}^i \widehat{Y}_{q_j}^{Est}}{i} - Y_q^{Act}}{Y_q^{Act}} \right| \times 100\%$$

where  $i \geq 1$ ,  $Y_q^{Act}$  is the actual subquery size,  $\widehat{Y}_{q_j}^{Est}$  is the estimated value derived from the  $j$ th sample trace. For an unbiased estimator, the absolute average relative error should approach 0 as  $i$  (the number of tries) increases [22].

As observed in Fig 5(c) and Fig 5(d), the errors on both TPCHIG05 and TPCHIG15 approached 0 as the number of estimations increased. The experiments validate the earlier proof of the unbiasedness of the sample trace estimator. Note that when the skew factor is relatively higher, in TPCHIG15, the relative errors are generally higher. Also, the error curves in Fig 5(d) converged smoother than those in Fig 5(c), where the sample ratios are equal to 1% and 0.1%, respectively.

## VII. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed utilizing information about how tuples are matching in the joins in a query, called the query trace, and estimate selectivities of all subqueries of a repetitive query. To gather sufficient information in the traces, we used the full outerjoins for queries with acyclic join graphs. We have shown that the exact selectivities of joins in all execution orders of the query can be computed from its trace.

To reduce the overheads incurred in collecting traces, we enhanced the outerjoins with sampling capability so that a random sample of the trace can be obtained efficiently. Experimental results have shown that with a small amount of sample tuples, accurate selectivity estimations can be obtained. Sample traces can be a very efficient and effective tool for the re-optimization of repetitive queries.

For future work, we would like to give an estimated sample size to get a satisfying estimation confidence interval with

efficient space budget. We are also interested in, for highly skewed data, how to use adaptive sampling on the relations to construct the adaptive sample trace.

## REFERENCES

- [1] S. Babu, P. Bizarro, and D. DeWitt, "Proactive re-optimization," SIGMOD '05, (New York, NY, USA), ACM, 2005, pp. 107–118.
- [2] N. Kabra and D. J. DeWitt, "Efficient mid-query re-optimization of sub-optimal query execution plans," SIGMOD '98, (New York, NY, USA), ACM, 1998, pp. 106–117.
- [3] V. Markl, V. Raman, D. Simmen, G. Lohman, H. Pirahesh, and M. Cilimdžić, "Robust query processing through progressive optimization," SIGMOD '04, 2004, pp. 659–670.
- [4] A. Aboulnaga and S. Chaudhuri, "Self-tuning histograms: building histograms without looking at data," SIGMOD Rec., vol. 28, June 1999, pp. 181–192.
- [5] S. Chaudhuri, V. Narasayya, and R. Ramamurthy, "Diagnosing estimation errors in page counts using execution feedback," (Washington, DC, USA), IEEE Computer Society, 2008, pp. 1013–1022.
- [6] L. Lim, M. Wang, and J. S. Vitter, "Sash: a self-adaptive histogram set for dynamically changing workloads," VLDB '2003, VLDB Endowment, 2003, pp. 369–380.
- [7] V. Markl, G. M. Lohman, and V. Raman, "LEO: An autonomic query optimizer for DB2," IBM Syst. J., vol. 42, Jan. 2003, pp. 98–106.
- [8] M. Stillger, G. M. Lohman, V. Markl, and M. Kandil, "LEO - DB2's learning optimizer," VLDB '01, (San Francisco, CA, USA), 2001, pp. 19–28.
- [9] F. Yu, W.-C. Hou, C. Luo, Q. Zhu, and D. Che, "Join selectivity re-estimation for repetitive queries in databases," in Database and Expert Systems Applications, vol. 6861 of Lecture Notes in Computer Science, pp. 420–427, 2011.
- [10] F. Yu, Constructing Accurate Synopses for Database Query Optimization and Re-optimization. PhD thesis, Carbondale, IL, USA, 2013.
- [11] K. Ono and G. M. Lohman, "Measuring the complexity of join enumeration in query optimization," VLDB '97, (San Francisco, CA, USA), 1990, pp. 314–325.
- [12] N. Bruno, S. Chaudhuri, and L. Gravano, "Stholes: a multidimensional workload-aware histogram," SIGMOD '01, (New York, NY, USA), ACM, 2001, pp. 211–222.
- [13] S. Christodoulakis, "Implications of certain assumptions in database performance evaluation," ACM Trans. Database Syst., vol. 9, June 1984, pp. 163–186.
- [14] C. M. Chen and N. Roussopoulos, "Adaptive selectivity estimation using query feedback," SIGMOD '94, (New York, NY, USA), ACM, 1994, pp. 161–172.
- [15] V. Poosala and Y. E. Ioannidis, "Selectivity estimation without the attribute value independence assumption," VLDB '97, 1997, pp. 486–495.
- [16] S. Chaudhuri, V. Narasayya, and R. Ramamurthy, "A pay-as-you-go framework for query execution feedback," Proc. VLDB Endow., vol. 1, Aug. 2008, pp. 1141–1152.
- [17] S. Chaudhuri and V. R. Narasayya, "An efficient cost-driven index selection tool for microsoft sql server," VLDB '97, (San Francisco, CA, USA), 1997, pp. 146–155.
- [18] G. Valentin, M. Zuliani, D. C. Zilio, G. Lohman, and A. Skelley, "DB2 advisor: An optimizer smart enough to recommend its own indexes," ICDE '00, (Washington, DC, USA), IEEE Computer Society, 2000, pp. 101–.
- [19] O. W. Paper, "Oracle coporation: Performance tuning using the sql access advisor." <http://otn.oracle.com>, 2003.
- [20] H. Herodotou and S. Babu, "Xplus: a sql-tuning-aware query optimizer," Proc. VLDB Endow., vol. 3, Sept. 2010, pp. 1149–1160.
- [21] B. Dageville, D. Das, K. Dias, K. Yagoub, M. Zait, and M. Ziauddin, "Automatic sql tuning in oracle 10g," VLDB '04, VLDB Endowment, 2004, pp. 1098–1109.
- [22] W. G. Cochran, Sampling Techniques, 3rd Edition. John Wiley, 1977.
- [23] S. Chaudhuri and V. Narasayya, "Program for tpc-d data generation with skew." <ftp://ftp.research.microsoft.com/users/viveknar/tpcdskew>.
- [24] M. Ley, "The DBLP computer science bibliography." <http://www.informatik.uni-trier.de/~ley/db/>.
- [25] J. Diederich, "FacetedDBLP." <http://dblp.l3s.de/dblp++.php>. [Retrieved Dec, 2013].

## APPENDIX

## A. Proof of Theorem 1

*Proof:* We show that there is a one-to-one correspondence between a result tuple of  $\pi_{R_{i_1}\text{-ID}, \dots, R_{i_m}\text{-ID}}T(P)$  and a result tuple of  $Q'$ .

" $\Rightarrow$ " By the construction of the trace, for each edge  $(R_i, R_j)$  in  $E$ , the  $R_i\text{-ID}$  and  $R_j\text{-ID}$  record the matches in  $T(P)$ . Let  $S$  be a set of tuples  $\{t_{i_1}, \dots, t_{i_m}\}$ ,  $t_{i_j} \in R_{i_j} \in V'$ , whose IDs appear in a result tuple of  $\pi_{R_{i_1}\text{-ID}, \dots, R_{i_m}\text{-ID}}T(P)$ . For each edge  $(R_{i_j}, R_{i_k}) \in E'$ ,  $1 \leq j, k \leq m$ ,  $t_{i_j}$  and  $t_{i_k}$  must match in their join attribute values because their IDs appear in the same trace tuple. That is, the set of tuples  $S$  satisfies all the join predicates placed between relations in  $V'$  and thus can generate a result tuple in  $Q'$  by joins. Moreover, the joins of  $\{t_{i_1}, \dots, t_{i_m}\}$  cannot generate more than one tuple.

" $\Leftarrow$ " Let  $S' = \{t'_{i_1}, \dots, t'_{i_m}\}$ ,  $t'_{i_j} \in R_{i_j} \in V'$  be a set of tuples that generates a result tuple in  $Q'$ . With joinable tuples from  $V - V'$  (they always exist because there is no dangling tuples in any join),  $S'$ , following plan  $P$ , can generate at least one result tuple in  $Q$ . Since  $S'$  has no null tuple, their corresponding IDs must appear as a result tuple of  $\pi_{R_{i_1}\text{-ID}, \dots, R_{i_m}\text{-ID}}T(P)$ . The projection  $\pi_{R_{i_1}\text{-ID}, \dots, R_{i_m}\text{-ID}}$  will retain only one set of IDs corresponding to  $\{t'_{i_1}, \dots, t'_{i_m}\}$  in  $\pi_{R_{i_1}\text{-ID}, \dots, R_{i_m}\text{-ID}}T(P)$ . ■

## B. Proof of Theorem 2

*Proof:* We show that there is a one-to-one correspondence between a result tuple of  $\pi_{R_{i_1}\text{-ID}, \dots, R_{i_m}\text{-ID}}T(P)$  and a result tuple of  $Q'$ .

" $\Rightarrow$ " Same arguments as in  $\Rightarrow$  of Theorem 1.

" $\Leftarrow$ " Let  $S' = \{t'_{i_1}, \dots, t'_{i_m}\}$ ,  $t'_{i_j} \in R_{i_j} \in V'$  be a set of tuples that generates a result tuple in  $Q'$ . With joinable tuples or null tuples (if there are no joinable tuples) from the relations  $V - V'$ ,  $S'$ , following plan  $P$ , can generate at least one result tuple in the query derived from  $Q$  in which all joins are replaced by full outer joins; and all these result tuples must have valid IDs for all  $R_i\text{-ID}$  attributes,  $R_i \in V'$ , since there is no null tuple in  $S'$ . Since duplicates are eliminated in  $\pi_{R_{i_1}\text{-ID}, \dots, R_{i_m}\text{-ID}}(T(P))$ ,  $S'$  can only produce one result tuple in  $\pi_{R_{i_1}\text{-ID}, \dots, R_{i_m}\text{-ID}}(T(P))$ . ■

## C. Proof of Theorem 3

*Proof:* First of all, we denote the query result size of  $q$  by  $Y_q$ . The total result tuples in the subquery  $q$  generated by the  $j$ th tuple of  $R_{i_1}$  is denoted by  $n_j$ . i.e. In relation  $R_{i_1}$ , its  $j$ th tuple generated  $n_j$  result tuples in the final result of subquery  $q$ . The query result size of  $q$  equals the sum of result tuples generated by each tuple on relation  $R_{i_1}$ . Thus  $Y_q = \sum_{j=1}^{|R_{i_1}|} n_j$ . Also note that  $S_{i_1}$  is a simple random sample without replacement of  $R_{i_1}$ . And  $n_{i_1}$  is the sum of result tuples generated by  $S_{i_1}$ . By Theorem 2.1 of [22], we have  $\hat{Y}_q$  is an unbiased estimator of  $Y_q$ . ■

# Efficient Aggregate Cache Revalidation in an In-Memory Column Store

Stephan Müller, Lars Butzmann, Hasso Plattner

Hasso Plattner Institute

University of Potsdam, Germany

Email: {stephan.mueller, lars.butzmann, hasso.plattner}@hpi.uni-potsdam.de

**Abstract**—Modern enterprise applications do not separate between online transactional processing and online analytical processing anymore. To ensure fast response times for expensive analytical queries, we implemented an aggregate cache in an columnar in-memory database. The separation into read-optimized main and write-optimized delta storage is exploited to cache only the aggregates based on the main storage and aggregate all records in the delta storage on-the-fly. This works with insert-only workloads, but not with deletes and updates that invalidate records in the main storage and consequently invalidate a cached aggregate. In this paper, we introduce an approach to revalidate a cached aggregate using efficient bit vector operations. A revalidation is superior to an invalidation since the old cached aggregate is reused. A further contribution is an evaluation of the influence factors that determine whether to invalidate or revalidate a cached aggregate. Our implementation shows that an aggregate cache revalidation outperforms an invalidation when less than 50% of the relevant records are invalidated.

**Keywords**-Aggregation, Materialized View Maintenance, In-Memory Data Base, Bit Vector

## I. INTRODUCTION

Several decades ago, database vendors decided to separate online transactional processing (OLTP) and online analytical processing (OLAP) systems due to performance issues. They created systems that were only optimized for a specific workload, either OLTP or OLAP. With improving hardware, e.g., multi-core CPUs and terabytes of main memory, database management systems (DBMS) are changing [1]. Further, today's enterprise applications have mixed workloads running both, transactional and analytical, workloads [2]. An example is the available-to-promise check (ATP). Stock movements in a warehouse represent transactional queries whereas the ATP check itself is an analytical query aggregating over the product movements to determine the earliest possible delivery date for a customer [3]. Other applications that require a DBMS to handle mixed workloads are the dunning process where the application determines which customers have outstanding payments and accounting applications that calculate the profit and loss statement based on the aggregated records.

One technique to speed up the execution of expensive analytical queries is the use of *materialized views* [4]. A view defines a function from a set of base tables to a derived table and is recomputed every time the view is referenced. A materialized view stores the result of a view in the database and therefore does not require a recomputation. All materialized views that contain an aggregation [5] in its definition are called

*aggregates* in this paper. Such pre-calculated results provide fast access to the data and reduce the overall load on the system. However, the benefit of speed comes with one tradeoff called *materialized view maintenance*.

Each time the underlying base tables are modified, a materialized view gets stale meaning the returned result is not up-to-date. The process of updating the materialized view in case of changes to the underlying base tables is called *materialized view maintenance*. This process was discussed in academia [6]–[8] and industry [9], [10]. However, the research is focused on data warehousing [8], [11], [12] and not on modern database architectures running mixed workloads. Compared to data warehouses, where maintenance downtimes may be feasible, transactional applications in mixed workload environments require high availability and throughput at any time.

In this paper, we introduce a mechanism to cache and revalidate analytical queries in the context of columnar in-memory databases (IMDBs). Columnar IMDBs got increased attention in the recent years since they are able to handle mixed workloads in a single system [1], [13]–[15]. Our work is based on [16] that introduced an aggregate cache for a columnar IMDB. The aggregate cache is a non-persistent caching engine inside the database. Like a materialized view, the aggregate cache consists of query results that are stored to speed up the access times. It leverages the main-delta architecture which separates a table into a read-optimized main storage and a write-optimized delta storage (cf. Figure 1). The idea of the aggregate cache is to cache only the main storage and unite the cached aggregate with the newly added records in the delta storage. This process is more efficient in many cases compared to calculating the complete result again. From the aggregate cache perspective, delete and update operations are equal and therefore called invalidations throughout this paper.

The aggregate cache guarantees that all results are up-to-date and it will never return a stale result. The aggregate cache consists of cache entries, each representing one unique aggregate query. A cache entry is created upon request and can be deleted upon request. Further, the cache entries are deleted in case the database shuts down or lacks main memory. For the future, we plan to include a mechanism into the cache that decides which queries are worth to cache and which are not.

For our evaluation, we have chosen a scenario of an ATP application. In our implementation, the application relies on a single, denormalized database table called Facts that contains all stock movements in a warehouse. Every movement consists

of an unique transaction identifier, the id of the product being moved, the date and the amount. The amount is positive if goods are put in the warehouse and negative if goods are removed from the warehouse. The aggregate that is based on the table Facts groups the stock movements by product and date and sums up the total amount per date and product. A detailed description of the ATP application can be found in our earlier work in [3]. We manually define the queries that will be cached and do not address the view selection problem [17] in the scope of this paper. We focus on the SUM and COUNT aggregation function as this is the dominant aggregate function in our introduced application. The aggregate cache also supports the standard SQL aggregate functions AVG, MIN, and MAX.

The paper is structured as follows: Section II gives a brief overview of related work in the area of materialized view maintenance. Section III explains the aggregate cache in detail including the algorithm and architecture. Section IV introduces the revalidation mechanism using the transaction manager and the incremental maintenance of a cached aggregate. In Section V, we analyze the cost factors of a revalidation. Section VI shows our experimental evaluation and Section VII concludes the paper with our main findings.

## II. RELATED WORK

Materialized view maintenance has been analyzed in academia [6]–[8] and industry [9], [10]. Blakely et al. were one of the first to propose a concept of incremental view maintenance [6] and Zhou et al. introduced a lazy view maintenance approach using delta tables [10]. These approaches are the foundation of our work. However, all work was done using traditional relational databases or even data warehouses with fewer restrictions [8], [11], [12]. In the context of this paper, we focus on enterprise applications consisting of OLTP and OLAP queries that require high availability and high throughput. The aggregate cache leverages the main-delta architecture that has not been evaluated before. To the best of our knowledge, an efficient maintenance strategy that is able to handle mixed workloads by leveraging available data structures of a columnar IMDB does not exist so far.

## III. AGGREGATE CACHE

The aggregate cache leverages the concept of the main-delta architecture as introduced in [16]. Separating a table into a main and delta storage has one main benefit. The separation allows to have a read-optimized main storage for faster scans and a write-optimized delta storage for high insert throughput. All inserts are inserted into the delta storage and are periodically propagated into the main storage in an operation called merge [18]. The fact that the main storage is only growing with a merge operation is leveraged by the aggregate cache so that only the results of the main storage are cached. All records from the delta storage are aggregated on-the-fly and united with the corresponding cached aggregate.

### A. Architecture

The aggregate cache is located inside the column store engine of SanssouciDB (cf. Figure 1). There is a single aggregate cache manager instance that manages all cache entries. A cache

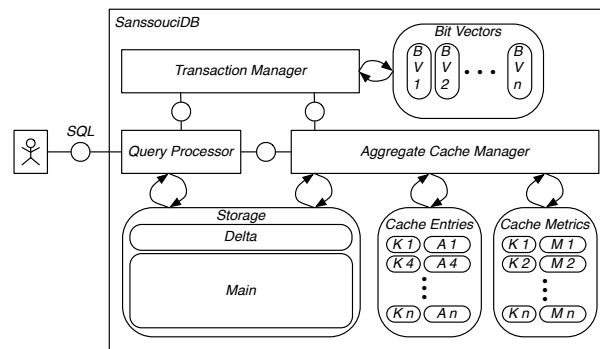


Figure 1. The internal architecture of SanssouciDB [2] with the main and delta storage, the aggregate cache manager and the transaction manager.

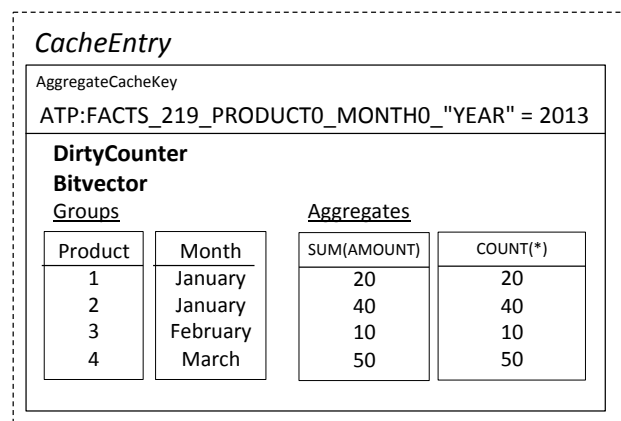


Figure 2. The structure of a cache entry for the ATP scenario.

entry consists of a key and a value. The key is composed of the table name, the group by columns, the where condition and the aggregates. The value is a complex structure consisting of the group by values and the corresponding aggregates. Additionally, relevant meta information about a cache entry, e.g. its creation time and its size, are stored in a structure called cache metrics. A detailed illustration is shown in Figure 2.

### B. Insert-only Scenario

Insert-only scenarios, where the application does not change any previously inserted data, are beneficial for the aggregate cache because of a non-changing main storage. There, deletes and updates are replaced by logical updates using differential values. This restriction can be a result of legal requirements or the business logic behind an application. An insert-only scenario never invalidates the cache and therefore uses its full potential. With each merge operation, the cache entries can either be incrementally maintained using the records from the delta storage or invalidated in case they are not used anymore. The decision whether to maintain or invalidate a cache entry can be made based on a combination of known cache replacement algorithms, e.g. LRU or LRFU, and the benefit and resource requirements of a cache entry.

### C. Cache Invalidation

Only a subset of applications have insert-only workloads. Most enterprise applications need to make changes to their

TABLE I. CONSOLIDATED BIT VECTORS OF TWO TRANSACTIONS RUNNING WITH TRANSACTION LEVEL SNAPSHOT ISOLATION.  $T_1$  HAS DELETED THE RECORD WITH ID 2.

Facts				Bit Vector	
Id	Product	Date	Amount	$T_1$	$T_2$
1	Tire	3/1/2013	10	1	1
2	Tire	3/7/2013	20	0	1
3	Brakes	3/3/2013	5	1	1
4	Tires	3/12/2013	30	1	1

data which affect the cache entries. According to an analysis of Krueger et al. [18], their analyzed customer workload consisted of 8% delete and update queries. In contrast, the TPC-C benchmark [19] consists of 35% delete and update queries. Even a minor change in the main storage where only a single record is updated or deleted invalidates the cache entries that are based on that record. An invalidation equals a complete recalculation of the result which is the least efficient way to handle such behavior. In this paper, we present a solution that is able to reuse the stale cache entries by extracting information from the transaction manager.

1) *Transaction Manager*: Transactions are a main functionality of a RDBMS. The transaction manager is responsible that the database has a consistent state after each transaction and ensures the ACID properties. One mechanism to implement concurrent transaction handling is multi version concurrency control (MVCC) [20]. Using MVCC, multiple transactions can run simultaneously and each have their own visibility on the database. In SanssouciDB (cf. Figure 1), the transaction manager creates a bit vector representing the visibility of a table for an incoming query based on its transaction token.

2) *Visibility Bit Vector*: Each table has four bit vectors, two for the main storage and two for the delta storage. For the aggregate cache, only the main storage bit vectors are relevant since a cache entry is only based on them. One bit vector contains the information about visible records (create bit vector) and the other bit vector the information about invalidated records (delete bit vector). The combination of both (using an exclusive OR) creates a bit vector which is called consolidated bit vector. It contains the actual visibility for a specific transaction. An example for two consolidated bit vectors is shown in Table I. Two transactions are running concurrently and query the table *Facts* with four records. In the beginning, all four records are visible to both transactions (consolidated bit vector 1111). Transaction  $T_1$  deletes the record with Id 2. Consequently, the bit at index 2 changes from visible to not visible for all further operations inside  $T_1$  (invalidated bit vector 0100 and consolidated bit vector 1011). Transaction  $T_2$  started at the same time as  $T_1$ . Depending on the isolation level,  $T_2$  can read the record with Id 2 (consolidated bit vector 1111 with transaction level snapshot isolation) or cannot read it after the delete (consolidated bit vector 1011 with statement level snapshot isolation).

#### IV. CACHE REVALIDATION

To prevent an invalidation of cached aggregates in case of invalidations in the main storage, the aggregate cache has to provide the functionality to calculate the modifications between the cache entry creation and its usage. Since the previously introduced transaction manager is responsible for

the visibility of records in a table, the aggregate cache can leverage that information to extract the invalidations.

##### A. Bit Vector Comparison

With each cache entry creation, a consolidated bit vector is stored as a snapshot of the database (cf. Figure 2). To further optimize the usage of a bit vector, only the bits after applying the WHERE clause of a query are used. We call these bits relevant bits. Additionally, a version counter called *dirty counter* is stored. The dirty counter is an integer value from the transaction manager that is incremented with each invalidation.

In Figure 3, the enhanced aggregate cache algorithm is displayed in detail. Compared to first version (without the grey colored boxes), the steps until the *AggCache Lookup* step are the same. In the next step, in case a cache entry was found, the *dirty counter* of the cache entry is compared with the current *dirty counter* of the table to determine if there has been an invalidation or not. In case the dirty counter has changed, the consolidated bit vector of the cache entry is compared to the current delete bit vector (retrieved from the transaction manager) using a bitwise AND to determine the relevant invalidations. The result is a bit vector containing all changes since the cache entry creation. If that bit vector has no bits set, e.g., the invalidated records did not affect the cache entry due to the filters in the WHERE clause, the cache entry is still up-to-date and nothing has to be done. If at least one bit is set, the aggregate query has to be executed on the main storage using only the bit vector containing the relevant invalidations. The output is a query result containing all the information that has to be subtracted from the cached aggregate to provide an up-to-date result. In IV-C, we explain how this information can be used for the incremental maintenance of cached aggregates and maintenance timings.

##### B. Bit Vector Compression

Storing a bit vector for each cache entry requires additional memory. To reduce the required amount, the aggregate cache can use different compression techniques. The characteristic of a bit vector is beneficial for compression due to its limitation of only two distinct values (0 and 1). If the characteristic of the application that is using the table is known, the aggregate cache can leverage that knowledge to choose the optimal compression techniques. Based on Abadi et al. [21], we propose three compression techniques that are most suitable for bit vectors. All techniques require to provide direct comparisons of two bit vectors without additional decompression overhead.

1) *Prefix/Suffix Encoding*: Prefix or suffix encoding is a simple compression technique where the first or last sequence of the same value is replaced by two single values, the value itself and its number of occurrences. For bit vectors, this technique is useful if the deleted values are at the beginning or end of the bit vector. Depending on the location of the deleted values, prefix or suffix encoding is applied.

2) *Run-length Encoding*: Like prefix encoding, run-length encoding replaces a sequence of the same value by two single values, the values itself and its number of occurrences. Unlike prefix encoding, this can be done for any sequence in the bit vector. This possibility makes run-length encoding

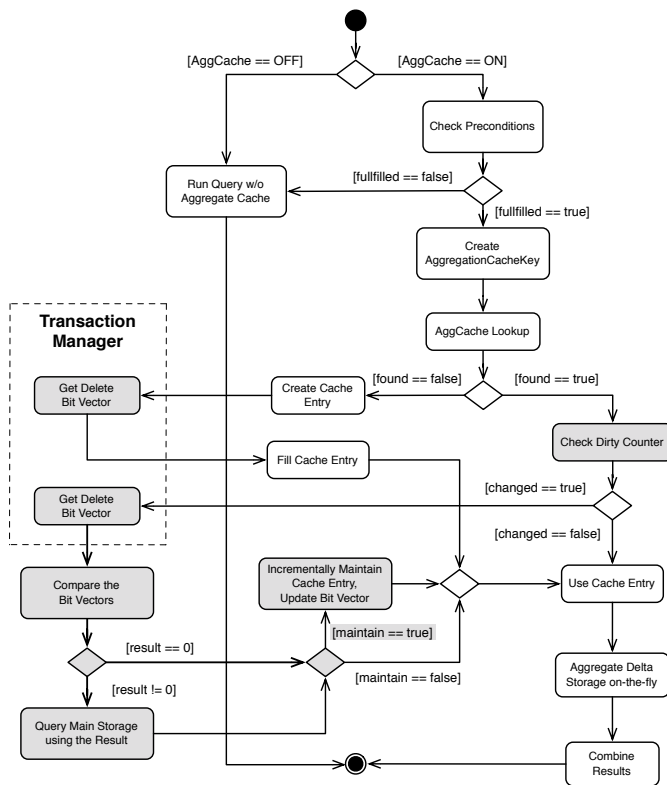


Figure 3. Activity diagram that visualizes the algorithm of the aggregate cache with the revalidation. The enhancements are colored in grey.

superior compared to prefix encoding. However, in case the number of deleted values is high and the bit vector shows the characteristic of an alternating pattern, the benefit of this technique decreases.

3) *Cluster Encoding*: Cluster encoding divides a bit vector into clusters of fixed size. Clusters that only contain equal values (only 0 or only 1) can be compressed and replaced by a single value. Clusters with mixed values (containing 0 and 1) are not replaced. The result is a new smaller vector. To remember which clusters have been compressed, an additional bit vector is created. Since the size of the cluster is not fix, it can be chosen based on the table size and the application characteristic.

The resulting compression ratio of each technique depends on the characteristic of the bit vector. A bit vector is influenced by the filter conditions in the WHERE clause that determines the ratio of relevant bits compared to the table size. Second, the application characteristic, e.g., are there bulk deletes or single deletes randomly distributed over the table. The automatic determination of the optimal compression technique based on the bit distribution is part of our future work.

C. Incremental Maintenance Timing

The maintenance of a cache is always a challenge. One crucial requirement for cache maintenance is to know what has changed. Using the introduced mechanism from Section IV-A, the aggregate cache is able to determine the modification that occurred between the cache entry creation and the current point

in time. Having that information enables the aggregate cache to incrementally maintain a cache entry. In the following, we introduce four different maintenance timings.

1) *Immediate Maintenance*: Immediate maintenance, also known as eager maintenance, maintains all cache entries with each invalidation in the main storage. A maintenance is done even if the cache entry is not accessed in the meantime. The benefit is that the cache entries are always up-to-date and do not require any additional calculation when the cache is accessed. The disadvantage is a significant overhead for workloads with higher insert ratios since more time for maintenance is required than a cache access can save. Additionally, the maintenance has to be done for all cache entries that are based on the modified table which increases the amount of maintenance with an increasing number of cache entries. If  $n$  cache entries are based on a table and one record is invalidated,  $n$  cache entries have to be maintained. In our previous work, we have evaluated this timing in the context of materialized views and showed that it is inferior to other maintenance timings [22].

2) *Deferred Maintenance*: Deferred maintenance, also known as lazy maintenance, maintains a cache entry with each cache access. As a result, a cache entry is up-to-date after each cache access. With this maintenance timing, the cost of a maintenance is with the actual cache access, and not with the delete or update. In their work, Zhou et al. [10] introduced lazy maintenance in the context of materialized views for Microsoft SQL Server, with the same motivation of shifting the costs towards the the view access. They also proposed to perform the maintenance in periods of lower loads to reduce the actual overhead during the view access.

3) *Periodical Maintenance*: Periodical maintenance is a lazy form of deferred maintenance. The maintenance can be based on different criteria. One criteria is a certain number of cache accesses. In that case, a cache entry is maintained every  $N$  cache accesses.  $N$  can be chosen depending on the workload characteristic. A second criteria is a certain threshold of the invalidation ratio. If that ratio is reached, a maintenance is performed. With a periodical maintenance, the cache entry is not up-to-date at all times, as shown in the scenario where an execution on the main storage is required with each cache access.

4) *Merge Operation Maintenance*: This strategy does not maintain the cache entry between two merge operations. After a cache entry creation, it always revalidates the cache entry using the additional execution on the main storage to return the correct query result. Due to the missing maintenance, this strategy should only be used for cache entries where the base data never gets invalidated. In that case, the additional main storage run is not required. A second use case for this strategy can be a less strict freshness of the cache entry. If an application only requires a rough estimation that can be based on 'older' data, this approach is sufficient. So far, we have not implemented a functionality to return old cache entries.

D. Merge Operation

The aggregate cache relies on the main-delta architecture. With the periodical merge operation, the delta storage is merged into the main storage and all deleted values in the main

storage are removed. The aggregate cache has to react to that change. There are three possibilities how the aggregate cache can react. Each cache entry can have its own strategy based on the decision of the aggregate cache. First, a cache entry can be invalidated and is removed from the aggregate cache. Second, a cache entry can be incrementally maintained. In that case, the delta storage has to be added to the cache entry and the invalidated records from the main storage have to be subtracted from the cache entry. Third, the old value of the cache entry is ignored and the aggregate query recalculates the cache entry after the merge operation has completed.

### E. Aggregate Functions

Aggregate functions can be divided into two categories, functions that are self-maintainable and functions that are not self-maintainable [23]. Looking at the standard aggregation function, SUM, COUNT, AVG (using SUM and COUNT) are self-maintainable with regards to inserts, deletes and updates. MIN and MAX are only self-maintainable with regards to inserts, but not with regards to deletes and updates because the information about the second highest/lowest value is not available. However, to overcome this issue, the database can store the  $n$  highest/lowest values, also known as MaxN/MinN. Using a data structure to store additional values enables an incremental maintenance until the data structure is empty and has to be refilled.

### F. Joins

Joins are a concept that combines records of multiple tables into one result. The combination is based on a common field, the join condition, among the tables. Even though the aggregate cache implementation does not support joins yet, we suggest a concept how the handling of joins can be done. Equally to the algorithm for a single table, a cache entry of a join query consists only of the result on the main storages. To return the complete result, three joins are required which are combined with the cache entry. First, both delta storages of the two tables have to be joined. Second, the delta storage of table A has to be joined with the main storage of table B. Third, the delta storage of table B has to be joined with the main storage of table A. The join between the two main storages is the most expensive join because most records are in the main storage (a ratio of  $>100:1$  [24]). The three other joins involve the smaller delta storages and therefore are cheaper.

Toerey et al. [25] introduced three types of relationships for relational databases: one-to-one, one-to-many, and many-to-many. The most frequently used type is the one-to-many relationship which is used to normalized database tables. For invalidations, the location of the invalidations is important. In case the invalidations only happen in one table, a single join is required to perform a maintenance. In case the invalidations happen in both tables (in a two table scenario) three joins are required. The third join makes sure that no values are removed twice, e.g., if records with the same join key from both tables are invalidated. In summary, a maintenance can vary between one join and three joins depending on the location of the invalidation. However, further knowledge about the application can reduce the complexity. An further implementation and evaluation has to verify this assumption and are part of the future work.

## V. COST ANALYSIS INVALIDATION VS. REVALIDATION

In most cases, a cache revalidation is beneficial compared to a complete recalculation because less data has to be accessed. However, a revalidation is not always the best solution, especially after large bulk deletes. Comparing the revalidation process with the recalculation process shows that the only difference is the input bit vector used for the aggregate query. The aggregate query itself, the query processing steps and the used data structures are the same. As a result, the only cost factor is the number of accessed records in the main storage. For a revalidation, it is the number of relevant invalidations in the main storage. For a recalculation, it is the number of relevant visible records in the main storage. For a decision in favor of an invalidation or revalidation, the aggregate cache has to determine the *InvalidationRatio* respectively *Benefit*. The *InvalidationRatio* describes the relation of relevant invalidated records to relevant visible records at cache creation (cf. Equation 1). If less than half of the records are invalidated, it is beneficial to revalidate the cache entry. If more than half of the records are invalidated, an invalidation performs better (cf. Equation 3). The parameter  $\alpha$  represents a factor which is required to combine the changes with the groups the cached result. Combining the two lists of results has a linear complexity. The factor will be further analyzed in Section VI-A.

$$InvalidationRatio = \frac{numSetBits(Bv_{visible} \wedge Bv_{delete})}{numSetBits(Bv_{visible})} \quad (1)$$

$$Benefit = \frac{1}{2} - InvalidationRatio - \alpha \quad (2)$$

$$Strategy(Benefit) = \begin{cases} \text{Revalidation} & Benefit \geq 0 \\ \text{Recalculation} & Benefit < 0 \end{cases} \quad (3)$$

The function *numSetBits* returns the number of set bits for a given bit vector.  $Bv_{visible}$  is the bit vector for relevant visible records at cache creation time, e.g. 00111011.  $Bv_{delete}$  is the current delete bit vector, e.g. 11100101. The bitwise AND would be 00100001 with 2 bits set. The resulting *InvalidationRatio* is 0.4 and consequently a revalidation is more beneficial.

## VI. EXPERIMENTAL EVALUATION

We implemented the aggregate cache in SanssouciDB but believe that the implementation in other columnar IMDBs with a main-delta architecture such as SAP HANA [26] will lead to similar results. Figure 1 illustrates the architecture of our implementation. The data and workloads we used are based on customer data and are parametrized to simulate different scenarios and patterns. The basic schema from our ATP scenario is shown in Table I with four columns.

All experiments and benchmarks have been conducted on a server featuring 8 CPUs (Intel Xeon E5450) with 3GHz and 12MB cache each. The entire machine was comprised of 64GB of main memory. Every benchmark in this section is run at least three times and the displayed results are the median of all runs.

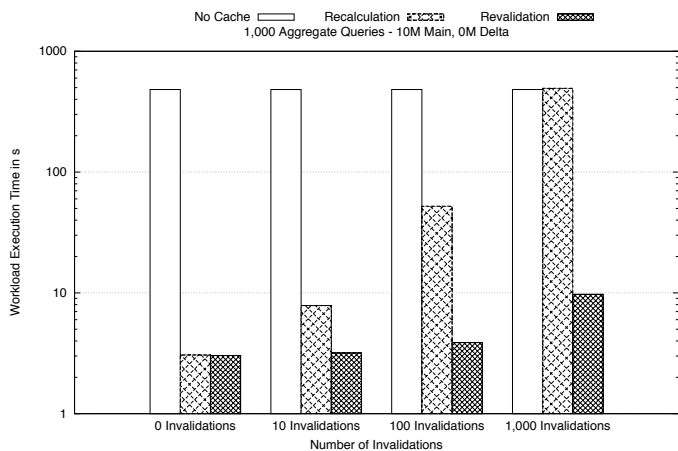


Figure 4. A revalidation outperforms an invalidation. The benefit of a revalidation increases which an increasing number of deletes.

### A. Invalidation vs. Revalidation

To show the benefit of a revalidation, we use four workloads with different invalidation behavior. A single OLAP query is executed 1,000 times. The different workloads vary in their number of invalidations. To have comparable execution times, the time for the delete operations will not be included into the measurement. The deferred maintenance timing is the default for all benchmarks. As seen in Figure 4, we compare the recalculation and revalidation strategy with the non-caching strategy to make the results more plausible. The non-caching strategy has equal query execution times since no caching is done and the OLAP query has to run 1,000 times. For the other two strategies, the query execution times vary depending on the invalidation behavior. With an increasing number of deletes, the difference between a revalidation and a recalculation increases, in favor for a revalidation. If a delete operation is always between two cache accesses (1,000 invalidations), a recalculation loses all its benefits compared to a non-caching strategy.

### B. Run-Time Analysis

The runtime of accessing a cached aggregate can be divided into four steps: a) Bit vector comparison b) Main storage access and aggregate calculation c) Maintenance d) Cache entry retrieval. All four steps are required for the revalidation strategy using a deferred maintenance timing. Steps 2 and 4 are used for the cache entry creation as well as cache entry recalculation (both are identical operations). The goal of this experiment is to measure the detailed costs for a cache entry creation respectively recalculation, and a cache entry revalidation. We also measure the costs for a cache access without a revalidation. A revalidation is measured using two scenarios with invalidation ratios of 1% and 10%.

Figure 5 shows the absolute results of the experiment with a logarithmic y-axis. The time to retrieve the cache entry is nearly the same for all. The maintenance costs for the revalidation strategy is also equal and not influenced by the invalidation ratio. Surprisingly, the comparison of bit vectors is so fast that the logarithmic y-axis cannot display it (the actual value 1ns). The main cost factor is the main storage access. For

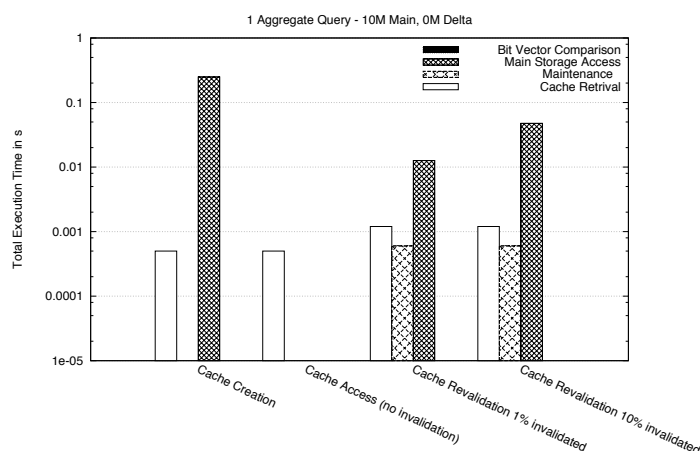


Figure 5. A benchmark that visualizes the run-time analysis of the different cache operations.

the revalidation, the time for accessing the main storage is the only time which increases with an increasing invalidation ratio. Comparing all four operations, the cache creation respectively recalculation is by far the most expensive operation.

### C. Invalidation Ratio

Figure 5 indicated that the performance of the revalidation depends on the number of invalidated records. This experiment measures the execution time of a revalidation for invalidations ratios ranging from 0% to 70%. The results are compared to a recalculation. Based on the introduced *InvalidationRatio* from Section V, we assume that both approaches are break-even (have the same performance) when the invalidation ratio is approximately 0.5. Further, we measure the influence of the result size using aggregate queries with 100 and 10k groups.

The experiment in Figure 6 confirms that the performance increases linearly with an increasing number of invalidations. The results also confirm the *Benefit* we have introduced with Equation 3. For small group size of 100, the ratio is approximately 0.5. At a ratio of 0.5 respectively 50%, a revalidation has the same performance as a recalculation. This point is also known as the break-even point. For the query with the larger group of 10k, the break-even point has shifted towards a smaller ratio. The performance of the revalidation is influenced by the size of groups since the revalidation has to match the groups of a cache entry to the groups of a revalidation. This process has linear complexity because each item of the group is accessed once (using a hash-based approach).

### D. Non-Relevant Invalidations

As explained in Section IV-A, the consolidated bit vector includes the information about relevant visible records. Figure 7 shows the benefit of that information in case non-relevant records are invalidated. Having this optimized bit vector, the aggregate cache is able to skip the main storage access in case the bit vector comparison produces a result containing only 0 (Figure 3). In contrast, the standard bit vector is not aware of that information and has to access the main storage until the query processor finds out that the result is empty. However,



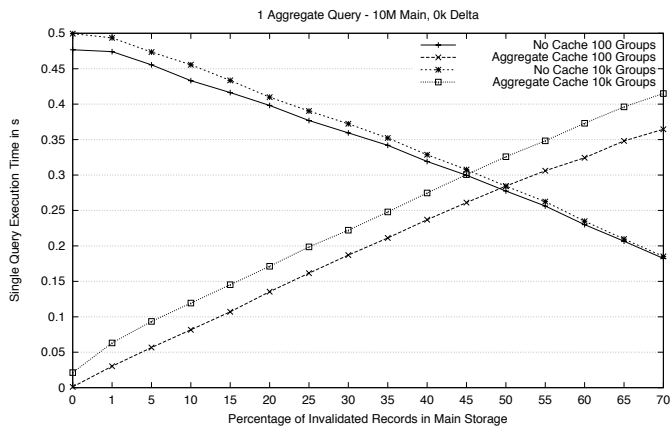


Figure 6. An experiment analyzing the influence of the number of invalidated records and the number of aggregate groupings on a revalidation.

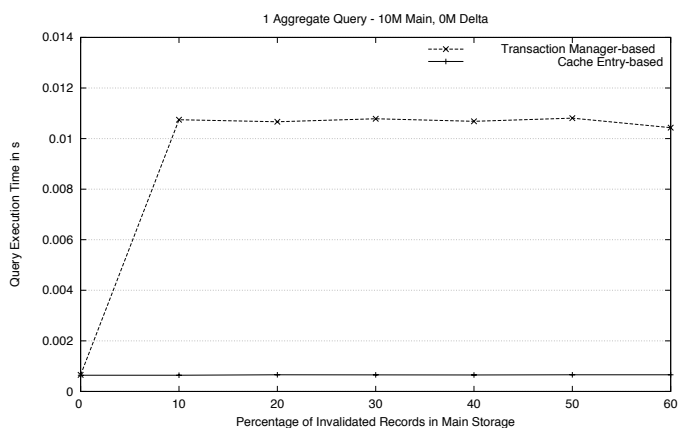


Figure 7. An experiment using on-relevant delete operations.

the experiment shows that despite an increasing number of invalidations on the main storage, the query execution time using the standard bit vector is constant.

### E. Maintenance Timing

In this paper, four maintenance timings for the aggregate cache are proposed (Section IV-C). Each timing has different maintenance costs that are influenced by three cost factors. In the following experiment, we focus on one cost factor. Based on the analysis of workloads we had access to, the ratio of invalidations to aggregate queries is the driving factor for the maintenance costs.

The experiment (cf. Figure 8) has five different aggregate queries which are executed 200 times each. The invalidations are distributed uniformly and always invalidate records that are part of the cache entries. The periodic timing has an  $n$  of 20 that revalidates a cache entry every 50 accesses.

As expected, all four timings have the same execution time in case no invalidations happen. For 10 invalidations, which occur every 100 aggregate queries, all timings create maintenance costs. The merge timing always has to access the main storage after the first invalidation, but does not maintain the cache entry. As a result, the costs are high, even though the

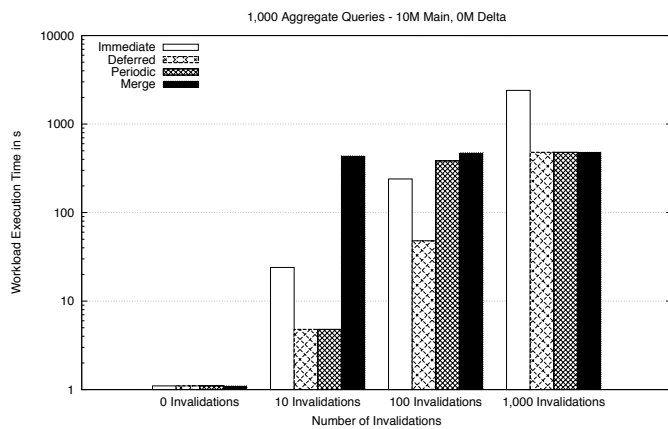


Figure 8. A comparison of the four proposed maintenance timings.

number of invalidations is low. The eager timing has to maintain five cache entries with every invalidation. This results in 50 revalidations. The deferred and periodic timing have lowest maintenance costs since they both require 10 revalidations. For the periodic timing, it is the best case since the revalidation of a cache entry is actually done right after the invalidation. With 100 invalidations (every 10 aggregate queries), the costs of the merge timing increase further. The costs for the eager and deferred time increase linearly. The periodic timing has a more than linear increase because the timing of the maintenance is not optimal. With 1,000 invalidations (one every aggregate query), the costs for the deferred, periodic and merge timing are the same. The eager timing has five times the maintenance costs of the other timings caused by the five cache entries. With only a single entry, the costs would be equally to the others.

In conclusion, the deferred maintenance timing is superior over the other timings. The costs of maintenance using an eager timing increase with an increasing number of cache entries. A periodic or merge timing might be applicable for workloads with a high number of invalidations, but the performance does never beat the deferred timing.

### F. Mixed Workload Benchmark

The CH-benCHmark created by Cole et al. [27] is a mixed workload benchmark combining the TPC-C and TPC-H benchmark. Based on their work and the available data generator, we created scenarios using three different scale factors (1, 10, 50). Since the aggregate cache does not support joins yet, we are only able to use five of the OLAP queries.

In Figure 9, the revalidation algorithm is compared with the recalculation and a non-caching strategy (for better comparability). A scale factor of 50 creates 60M records in the order line table. The number of queries was fix for all three scale factors. The workload contains 3561 inserts, 230 updates, and a total of 380 aggregate queries. Each of the five aggregate queries was executed 76 times. The revalidation algorithm uses the lazy maintenance timing as this is the best performing timing (cf. Figure 8).

The results show that a revalidation outperforms a recalculation for all scale factors. A revalidation is up to 5

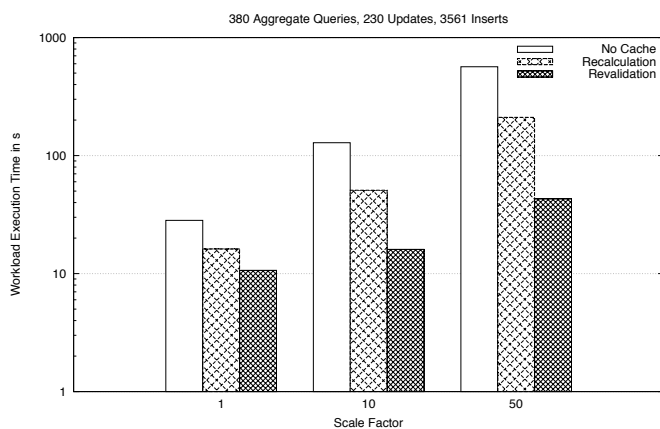


Figure 9. The CH-benCHmark [27] results.

times faster (scale factor 50). The results indicate that the performance advantage increases with an increasing scale factor. This validates the efficiency of the proposed revalidation algorithm.

## VII. CONCLUSION

In this paper, we have introduced an efficient strategy to revalidate cached aggregates in an in-memory column store. Based on the existing insert-only implementation, we have explained the idea of the aggregate cache and motivated the necessity of a cache revalidation strategy for mixed workloads with updates and deletes. Using the available information from the transaction manager, the aggregate cache is able extract the information of invalidated records. A bit vector containing snapshot information of the database is added to the cache entry. To keep the cache entry size as small as possible, we have proposed three compression techniques that reduce the required amount of memory for a bit vector. We have compared the process of a recalculation with a revalidation and created a cost function to determine the optimal decision between the two. In our evaluation using an ATP scenario consisting of transactional as well as analytical queries, the experiments reveal that a revalidation outperforms an recalculation up to an invalidation ratio of 50%. The influencing factor of the revalidation is the number of relevant invalidated records. Our optimization to include the filter conditions into the bit vector reduce the amount of maintenance significantly.

In our future work, we plan to include the support for joins into the aggregate cache. Existing admission and evictions strategies can be extended by run-time information of the aggregate cache manager, for example the execution time or the result size. Also, an evaluation of our algorithm using other database architectures without a main-delta architecture is subject to further research.

## REFERENCES

- [1] H. Plattner, "A common database approach for oltp and olap using an in-memory column database," in SIGMOD, 2009, pp. 1–2.
- [2] —, "Sanssoucidb: An in-memory database for processing enterprise workloads," in BTW, 2011, pp. 2–21.
- [3] C. Tinnefeld, S. Müller, H. Kaltegärtner, S. Hillig, L. Butzmann, D. Eickhoff, S. Klauk, D. Taschik, B. Wagner, O. Xylander, A. Zeier, H. Plattner, and C. Tosun, "Available-to-promise on an in-memory column store," in BTW, 2011, pp. 667–686.
- [4] D. Srivastava, S. Dar, H. Jagadish, and A. Levy, "Answering queries with aggregation using views," in VLDB, 1996.
- [5] J. M. Smith and D. C. P. Smith, "Database abstractions: Aggregation," Commun. ACM 1977.
- [6] J. A. Blakeley, P.-A. Larson, and F. W. Tompa, "Efficiently updating materialized views," in SIGMOD, 1986, pp. 61–71.
- [7] A. Gupta and I. S. Mumick, "Maintenance of materialized views: Problems, techniques, and applications," IEEE Data Eng. Bull. 1995.
- [8] D. Agrawal, A. El Abbadi, A. Singh, and T. Yurek, "Efficient view maintenance at data warehouses," in SIGMOD, 1997.
- [9] R. G. Bello, K. Dias, A. Downing, J. J. F. Jr., J. L. Finnerty, W. D. Norcott, H. Sun, A. Witkowski, and M. Ziauddin, "Materialized views in oracle," in VLDB, 1998, pp. 659–664.
- [10] J. Zhou, P.-A. Larson, and H. G. Elmongui, "Lazy maintenance of materialized views," in VLDB, 2007, pp. 231–242.
- [11] Y. Zhuge, H. García-Molina, J. Hammer, and J. Widom, "View maintenance in a warehousing environment," in SIGMOD, 1995, pp. 316–327.
- [12] H. Jain and A. Gosain, "A comprehensive study of view maintenance approaches in data warehousing evolution," SIGSOFT Softw. Eng. Notes 2012.
- [13] M. Grund, J. Krüger, H. Plattner, A. Zeier, P. Cudre-Mauroux, and S. Madden, "Hyris: a main memory hybrid storage engine," in VLDB, 2010, pp. 105–116.
- [14] A. Kemper, T. Neumann, F. F. Informatik, T. U. Mnchen, and D-Garching, "Hyper: A hybrid oltp&olap main memory database system based on virtual memory snapshots," in ICDE, 2011.
- [15] S. Idreos, F. Groffen, N. Nes, S. Manegold, K. S. Mullender, and M. L. Kersten, "Monetdb: Two decades of research in column-oriented database architectures," IEEE Data Eng. Bull. 2012.
- [16] S. Müller and H. Plattner, "Aggregates caching in columnar in-memory databases," in 1st International Workshop on In-Memory Data Management and Analytics (IMDM), in conjunction with VLDB 2013, Riva del Garda, Trento, Italy, 2013.
- [17] H. Gupta, "Selection of views to materialize in a data warehouse," in ICDDT, 1997.
- [18] J. Krueger, C. Kim, M. Grund, N. Satish, D. Schwalb, J. Chhugani, H. Plattner, P. Dubey, and A. Zeier, "Fast Updates on Read-Optimized Databases Using Multi-Core CPUs," in VLDB, 2012.
- [19] F. Raab, "TPC-C - the standard benchmark for online transaction processing (OLTP)," in The Benchmark Handbook, 1993.
- [20] P. A. Bernstein and N. Goodman, "Concurrency control in distributed database systems," ACM Comput. Surv., vol. 13, no. 2, Jun. 1981, pp. 185–221.
- [21] D. Abadi, S. Madden, and M. Ferreira, "Integrating compression and execution in column-oriented database systems," in Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, ser. SIGMOD '06. New York, NY, USA: ACM, 2006, pp. 671–682.
- [22] S. Müller, L. Butzmann, K. Howelmeyer, S. Klauk, and H. Plattner, "Efficient view maintenance for enterprise applications in columnar in-memory databases," in EDOC, 2013, pp. 249–258.
- [23] I. S. Mumick, D. Quass, and B. S. Mumick, "Maintenance of data cubes and summary tables in a warehouse," in SIGMOD, 1997.
- [24] H. Plattner and A. Zeier, In-memory data management: an inflection point for enterprise applications. Springer-Verlag Berlin Heidelberg, 2011.
- [25] T. J. Teorey, D. Yang, and J. P. Fry, "A logical design methodology for relational databases using the extended entity-relationship model," ACM Comput. Surv., vol. 18, no. 2, Jun. 1986, pp. 197–222.
- [26] F. Färber, S. K. Cha, J. Primsch, C. Bornhövd, S. Sigg, and W. Lehner, "SAP HANA database: data management for modern business applications," SIGMOD, 2011.
- [27] R. Cole, F. Funke, L. Giakoumakis, W. Guy, A. Kemper, S. Krompass, H. Kuno, R. Nambiar, T. Neumann, M. Poess, K.-U. Sattler, M. Seibold, E. Simon, and F. Waas, "The mixed workload ch-benchmark," in Proceedings of the Fourth International Workshop on Testing Database Systems, ser. DBTest '11. New York, NY, USA: ACM, 2011, pp. 8:1–8:6.

# Enterprise Data Solution Leveraging Data Warehousing for Big Data Veracity at Saudi Aramco

Muhammad Shehryar Khakwani  
Upstream Database Services Division  
Saudi Aramco  
Dhahran, Saudi Arabia  
e-mail: muhammad.khakwani@aramco.com

**Abstract**— This paper deals with the challenge faced by large organizations of establishing Big Data veracity, for maximizing their return on investment. Large enterprises, whether commercial or government, are collecting data at an unprecedented rate. Nowadays, data entry is not limited to traditional back office transactions; increasingly, data is gathered from sensors installed in physical assets and transmitted in real-time over large networks. Data technology professionals approach this new wealth of data at various levels defining innovative techniques and applying them in areas of data storage, data warehousing, data mining, semantic logic algorithms, complex event processing etc. For business stakeholders, the benefit is not in the increasing speeds of data acquisition, nor in the variety of data gathered. Return on investment is realized when the data is transformed into information which decision-makers can trust for making operational and analytical decisions, giving the organization a competitive advantage. Data Warehousing plays a critical role in assessing Big Data veracity, identifying problem areas, and building the trust needed for decision-making.

**Keywords**- Big Data; Veracity, Data Warehouse; Oil and Gas.

## I. ENTERPRISE DATABASES

Enterprises today retain more data than they did a decade ago. Databases supporting large enterprises no longer obtain data only from traditional back office data entry systems. Businesses realized the competitive advantage that can be provided by useful and timely information, and fuelled the drive for building efficient data gathering systems. These systems succeeded in streamlining and automating data gathering, in many cases eliminating manual data entry, providing businesses quicker data access, thereby giving rise to Big Data. Gartner Inc. defines Big Data as “high-volume, high-velocity and high-variety information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making [1].”

Enterprises moved towards a host of best-of-breed technologies, adopting ones best suited to address a particular need, and in their wake left large volumes of data, in varying formats, arriving at various frequencies. The resulting complexity creates new data management issues for

businesses to deal with today. More companies than ever want practical application of Big Data concepts to data sets which are too large to handle with traditional systems [2]. Businesses serious about successfully extracting value from Big Data initiatives will no longer have the luxury of treating data stores with outdated data management policies. Successful enterprises will incorporate policies and practices for dealing with Big Data as part of their comprehensive data governance strategies.

This paper is based on real occurrences in the industry. It describes the situation at Saudi Aramco before Big Data, the introduction of real-time data gathering which resulted in the arrival of Big Data, the problem posed by data veracity, and the critical role Data Warehousing played in dealing with it. It concludes with the use of data warehousing which provides a feasible and implementable solution towards establishing veracity for Big Data.

## II. BIG DATA VERACITY CURRENT SITUATION

The industry today is struggling with Big Data for its volume and velocity, and veracity has emerged more recently. It is only recently that the big data is characterized not only along the three “V”s, volume, variety and velocity, but also along a fourth “V”, veracity [3]. IBM recognizes this as establishing trust presenting a huge challenge as the variety and numbers of source grows [4].

## III. SAUDI ARAMCO UPSTREAM DATA

Saudi Aramco has produced oil since the 1930s, and is currently the world’s prevalent oil producer. It remains committed to serving the world’s energy needs, and maintaining its standing among the leading energy companies. In order to achieve its objectives, conduct safe operations, and optimize production strategies, Saudi Aramco committed to investing in Information Technology several decades ago.

Production and Reservoir engineers have been gathering and analyzing scientific data for decades at Saudi Aramco. Data acquisition can be categorized in two broad categories, traditional data gathering, and more recently, real-time streaming data gathering.

### A. Traditional Data Gathering

Oil companies have collected data to manage their upstream assets, and Saudi Aramco has been on the forefront of such efforts. Engineers require geo-scientific data such as fluid production rates, well and formation pressure readings, and rock properties to make decisions for harvesting hydrocarbons properly. This data is typically acquired by production engineers conducting operations on wells in oilfields, using various gauges to record measurements. The data gathered during operations is entered in a master database, where it is verified for accuracy by ranking engineers. An engineer compares the readings taken to previous measurements, checks nearby wells, and takes into account subsurface factors before approving the values for use.

These processes for gathering and validating data have been in place for years, and have matured over time. Tried and tested, dependable systems upload and retrieve data. Consequently, the trust, or for lack of a better term, *feel* for these values, has also been intrinsically built over the years.

Geoscientists and decision-makers who rely on geo-scientific data to make critical decisions feel far more comfortable when the data they are considering has been reviewed, validated, and signed-off by a qualified engineer. This is especially true when dealing with scientific data related to subsurface measurements; earth can be quite finicky to deal with at the best of times!

This sentiment of having trust in numbers before taking appropriate action is not surprising; I myself trust medical lab test results for my annual checkup after my doctor has looked them over, considered similar tests from years past, my condition at the time, and given his approval for the lab work.

The key point here is that trusting data quality, accuracy, reliability, i.e., its *veracity*, is built into processes for traditional data gathering.

### B. Real-Time Streaming Data Gathering

Real-time monitoring of well performance is a complex, multi-departmental and very expensive undertaking at Saudi Aramco [5]. Real-time data gathering in the digital oilfield is based on the premise that getting information faster will alert to potential problems earlier, enable timely intervention, thereby saving costs, and inevitably resulting in safer operations.

For oil and gas producers, real-time streaming data is gathered during drilling and producing operations. Sensors may be installed on surface, for example on pipelines to measure the rate of fluid passing through it, or installed deep inside oil and gas wells to check subsurface temperature and pressure at reservoir conditions. These sensors provide a constant data feed for the ever-changing big picture of a modern day digital oilfield.

This real-time, or *near* real-time, monitoring of assets generated a lot of interest within the user community. There was a real push from the business to unlock the potential from getting information faster.

## IV. DIGITAL OILFIELDS GENERATE BIG DATA

Real-time data is transmitted from oilfields as a continual stream, and consumed by software systems for alerting, reporting, analysis, and ideally providing a visual up-to-the-minute representation. As more and more oilfields were equipped with sensors, data started to flow in steadier, stronger streams; Big Data had arrived. Tens of thousands of readings started streaming in from sensors installed across thousands of assets.

### A. Determining Data Properties

To establish whether indeed it is Big Data, let us examine the properties of incoming data. The data is machine generated and arrives as key-value pairs at a very high frequency, so it has high *velocity*. The numerous types of sensors transmit different readings such as reservoir pressure, flow rates, temperature etc., so it has a diverse *variety* of data types. At Saudi Aramco, this activity certainly generates an increasingly large *volume* of data.

### B. Determining Critical Success Factors

Projects which introduce new technology, with an aim to transform traditional business practices, normally encounter unanticipated problems. Success requires recognizing problems early, and reacting effectively. This holds true when executing Big Data projects in larger, mature enterprises. Unforeseen challenges will need to be addressed during project execution.

#### 1) Technical Factors

For Information Technology (IT) professionals, focus quickly shifts to IT related concerns like calculating storage capacity for incoming streams, efficiently indexing key-value pairs, and minimizing retrieval time for display on large monitors. While these and other activities like procuring the right hardware, accurately estimating and accounting for scaling volumes of data, configuring the systems to optimally run with low latency, are all important factors, they do not, by themselves, declare a successful Big Data implementation.

#### 2) Business Factors

It is vital that success factors be defined from a business perspective. Our measure of success is whether the engineers and geoscientists are able to utilize the information, transform their processes, conduct better analyses, and add value to the bottom line.

### C. Problems Encountered

The first set of problems to overcome tended to be technical in nature. The networks needed to be extended to reach all the assets; oilfields tend not be in the most hospitable areas. The bandwidth capacity needed to be upgraded to handle large volumes of incoming data, and clusters of servers procured to receive and store the data.

Careful capacity planning, savvy cost estimates, and forthright communication with management to secure funding are essential parts of providing a satisfactory technical infrastructure. These are reasonably predictable issues, and experienced project managers are able to address these given adequate resources.

Where Big Data projects differ are when dealing with uncertainty and dealing with earth sciences compounds the problem. Uncertainty is an undeniable reality when dealing with Big Data. Managing uncertainty and establishing trust are key to extracting value from Big Data [3].

The problem which proved to be critical turned out to be an intangible one. The single most important factor that lies between success and failure, and the hardest to achieve, turns out to be data veracity: *establishing trust*.

## V. PROBLEMS DETERMINING VERACITY IN DATA

It soon became evident that real-time data could not be treated as “the same thing a lot faster”. Though Big Data was similar in its look and feel to conventional data, it needed to be treated differently.

Traditional data collection allows engineers to study and validate the data, determine its usefulness and only keep the most reliable data. Those using the data trust its quality and base their decisions on it. It is not possible to give the same level of care to high velocity data. It is unreasonable to continue to try and monitor that amount of data using manual processes.

Following are a few select examples from real life which show the different types of problems encountered when dealing with data veracity in real-time:

### A. Validity Domains and Ranges

One of the first data validation actions when dealing with data is to establish validity ranges for incoming values. Valid value domains, and ranges, were determined after studying existing patterns of data gathered traditionally. Subject matter experts were consulted and acceptable values defined. For example, fluid pressure readings should lie within 1,500 to 4,000 psi for flowing wells.

In reality, it was never as simple as an absolute range. What the users really wanted were ranges which were put in context after considering various factors. For instance, consider the geographical locations of wells, take into account the preceding 24 hours, examine some other relevant factors and then apply a range of values for determining validity. These were all the checks that the engineer carried out when taking readings manually.

For Big Data this can be problematic since the *volume* of data arrival can prevent extensive checking – if everything is not checked very quickly, a lot more data is waiting in the queue. At a technical level one has to be very careful in writing extremely efficient code, and make smart choices about what can be verified in a very short period of time.

### B. Good Spikes Bad Spikes

In general values that are out of a given range are excluded when reporting. The sensors are installed in harsh conditions, above and below the surface; they may need to be re-calibrated etc. However, what may look erroneous at first, may turn out to be correct. While conducting a data review for streaming data with senior field engineers, a set of wells was chosen and data analyzed. An analyst noticed some spikes in production rates and pointed them out. The

field engineers asked for the time of day when the spikes occurred, and then mentioned that the wells during that time were being adjusted and tested for choke settings which would have caused these spikes in rates. A choke is a heavy steel nipple inserted in production tubing used to restrict flow and control pressure [6]. Had these readings been taken through traditional methods, the engineer taking the readings would have been aware of the operations being conducted and taken that into account when validating the data.

Clearly there is a dependency between production rates and choke settings on a well. If the well has sensors for choke settings, then there is a dependency between streaming data readings. If the well is sending production rates only, then engineers must provide the data for choke settings using traditional gathering means. However, by the time data is entered by the engineer using traditional methods, it may be too late for verifying the streaming high *velocity* data.

### C. Conflicting Values

In situations, similar to the ones mentioned above, where there is a dependency, applying verification rules can be problematic. There are several instances of inter-dependent streaming values. Consider two sensors, one indicating whether the well is flowing, and another measuring the fluid rate. The majority of the time these will be consistent. There are cases when one indicates the well is flowing, and the other shows a production rate of zero. Which one is now correct? Which sensor value do you trust?

For a decision-maker who wants to determine the volume of fluid produced and take appropriate action, this poses a confidence problem. Increasing or decreasing volumes of fluid require different actions. Conflicting results immediately cause a loss of confidence in the entire data set. Yes, there are techniques where running mathematical algorithms, or checking a third or fourth set of variables determines the veracity among the conflicting variables, but decision-makers hesitate before authorizing expensive action if results are murky. This is understandable especially where expensive decisions are at stake. Consider this: would you make an investment and buy a stock if you were told the streaming price you see is 90% correct, only a 10% chance that it is not the actual stock price?

### D. Suspiciously Correct

Another case which crops up when dealing with Big Data shows values that seem perfectly valid, fall within defined validity ranges, but raise suspicion. These may come from a sensor which is transmitting the same value for a period of time, or the value is within a very narrow range over an extended period. Typically this unsettles the data consumer, because he is not sure whether the sensor is malfunctioning or ‘stuck’ and transmitting the same pattern repeatedly. It is hard to distinguish if the fluid rate really did remain constant, or whether the sensor transmitting data malfunctioned during the period of time. If the values are repetitive for a short period, that may be acceptable, but if it is absolutely constant for a longer period of time then something is likely wrong. When determining veracity, it is difficult to determine exactly when incorrect readings began being transmitted?

How far back in time does the problem go with readings from a sensor, and whether to fix the data or discard the readings?

*E. Multiple Truths*

As strange as this statement may sound, real life deals with multiple truths. When dealing with the scale of enterprise level data, the Big Data picture is usually composed of smaller pieces of information obtained from different data stores. Resolving these differences does not necessarily identify only one trusted source.

For instance, analyzing data elements from a well, and cross checking against spreadsheets and charts recorded by engineers, does not show one source always being correct over the other. There are cases where the sensors showed conditions which the engineers missed, and vice versa where the engineers had more accurate information and better observations.

In one case, while visiting engineers in the field for this project, sensors were transmitting very low fluid rates. The field engineer saw this and instructed the values be ignored, since the well had been shut-in the previous day, and therefore the fluid flow rate should be recorded as zero. In other cases, sensors showed more accurate choke setting information than data entered manually.

Clearly establishing criteria for acceptance is the key to data veracity. In real life, especially when dealing with large databases which input data from a *variety* of sources, defining a comprehensive set of rules which can always be applied programmatically is not straightforward.

VI. DATA WAREHOUSING—CRITICAL FIRST STEP

Large enterprises deal with various facets of data management simultaneously. Problems do not lie neatly within a single Data Governance domain such as Data Ownership, Data Quality, Data Mining, Data Integration, or High-volume Streaming. An enterprise solution must draw on various disciplines within data management to provide an effective solution. For a large enterprise like Saudi Aramco, spread out over a vast geographical area, with physical and data assets handled by multiple organizations, the challenge was to gain an effective overall understanding.

Data warehousing provides an all-inclusive view of data. The introduction of a data warehouse into the Big Data architecture fulfilled a critical need to bring disparate data sets together in order to establish veracity.

Data warehousing provides an overall understanding of data associations by placing facts together and along a consistent time dimension. A data mart which pulls data from various sources and lets the users check for accuracy goes a long way towards dealing with data veracity. Fig. 1 shows a conceptual view of a data warehouse bringing together real-time and traditionally collected mastered databases to provide a unified view.

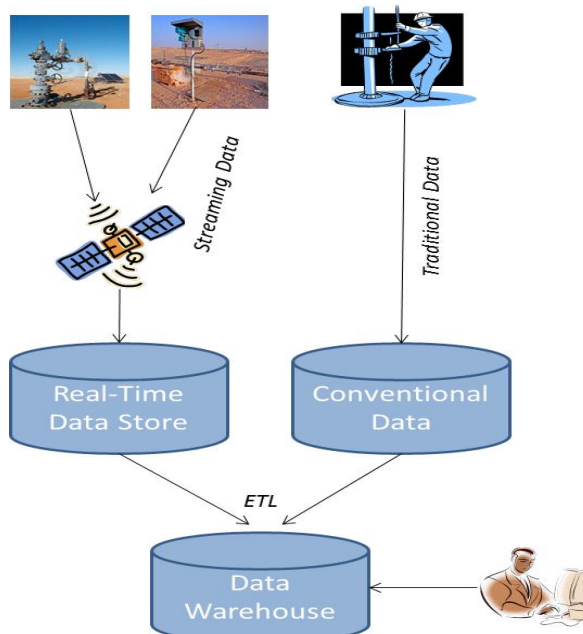


Figure 1. Consolidate data sources for understanding

A. Data Transformation: Smoothing Out Time

At first look, data warehousing seems exactly the wrong approach to the problem. Data warehouses are centered on data cubes with well-defined dimensions, where time plays a crucial role in setting up a data warehouse. When dealing with big projects generating big data arriving from multiple data sources, the time grain is completely askew because the *velocity* of the two data sources (conventional and real-time) is so different. In large industrial applications, data origins spanning multiple organizations cannot be taken in isolation. The two data sets need to be considered to supply context, from a user’s perspective.

The time problem is tackled by agreeing on a common baseline for time on the fact table. In our case, it turned out to be a day since users wanted to construct a daily picture for each well. These meant taking readings coming in at various intervals; some arriving every few minutes, others ever few seconds, and aggregating them into a daily summary in the data warehouse.

A key point here is to include only those factors that establish *veracity* and are utilized for analysis. It would be exhaustive to do this for every type of reading. For instance only flow rates, pressures and choke settings were considered, and many other readings like pump vibrations and temperature sensor readings were ignored. Many data elements do not add value to the process of establishing veracity. It is very important to consult subject matter experts from users for determining what is important to gain their confidence – *remember the end goal is to have them use this for better decision-making.*

We defined our fact table to contain information for every calendar day, complementing each streaming value aggregate with the latest values from the master data store, and essentially constructed a “big picture of big data”.

### B. Advantages of Data Warehouse

There are several advantages to introducing a data warehouse in the architecture and leveraging it to gain a better understanding of Big Data. These include:

- **Consolidating Information:** The advantage of a Relational Online Analytical (ROLAP) design is that it puts the relevant data *on one row* in the fact table. The Extract Transform Load (ETL) which populates the data warehouse processes data for each well, every day, consolidating data collected from sensors displayed side by side with the latest values available from traditional data sources.
- **Consistent Timeline:** When ETL constructs a daily picture, it solves a major problem of what to do with data collected over time using different methods. Enterprises that have collected data over years, and choose to modernize sanctioning mega-projects using the latest technology cannot simply discard the values they obtained earlier. All valid data collected over time must be made available to users. The data warehouse can construct data from when it first becomes available in the traditional data store, and continue to add information from when the wells are instrumented to provide streaming data.
- **Consolidating Sources:** Once the data is available in a data warehouse, it becomes much easier to spot data anomalies. Data from different sources, collected by multiple professionals, and under the ownership of various organizations is now available in one place to look over. This goes a long way towards establishing data veracity.
- **Perform Analysis:** Having the data in a data warehouse lends itself naturally to running analytical queries. Aggregated values from streaming data which seem suspect are easily picked up, and when necessary one can drill down to the actual readings and analyze a manageable set of information. For instance, it is easy to run SQL queries and check for pressure drops over a threshold, say 5 or 10% in daily averages and look closely at the days where data patterns do not align. The advantage gained is that while one can scan real-time data stores, base lining and transforming data into a warehouse makes it far more manageable. This makes it easier to scan for missing data, and search for patterns where data does not fit with the rest of the readings.

### C. Visual Representation

If a picture is worth a thousand words, then visualization must be worth at least a thousand data points. There are several tools available which allow quick visualization using charts and graphs to analyze data in data warehouses. Having the data in one unified data source makes it possible for applications to perform better data analysis and generate comprehensive reports readily. Reporting tools do not have to run exclusively either on streaming, or master data stores, placing the onus on the user to merge the results. Instead reports are built on a correlated data set, stored in fact tables

in a data warehousing structure, and presented as a unified view to the end user. Fig. 2 shows a visual representation of streaming data (shown in green, and traditionally gathered data shown in red) placing them next to each other. This makes it easier to identify the outlier scattered values, and those that should be considered.

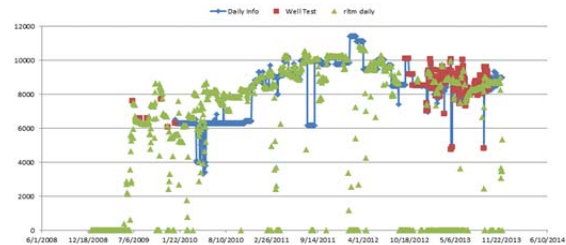


Figure 2. Plotting Traditional and Streaming Data[7]

## VII. FIXING PROBLEMS SHOWN BY WAREHOUSING

Data warehousing will make it easier to identify, but not resolve the data issues. Business does not get the intended return on investment until the underlying problems are fixed. When dealing with Big Data, getting the data actually fixed turned out to have its own peculiarities. Some of these characteristics of Big Data fixes include:

### A. Technical Challenges

Applying a data fix on voluminous large data sets poses technical problems. Unlike traditional data where incorrect values are updated easily to reflect the correct value, typically, one does not change or update streaming data values generated from sensors, even if they are wrong. They are left alone; instead validation or logic is applied to determine which values to ignore. Sometimes even the validation logic is modified. It does not happen often, but occasionally, engineers do change their minds. If at first they allowed pressure values that fell within a range of 3,000 to 3,500 psi, they may ask us to relax validation rules to include values up to 4,000. In this case, we needed to back up and re-process an awful lot of data. Such re-processing, or re-summarization, crunches through a lot of numbers, and sometimes takes days to complete. In such cases, process data fixes going backwards in time, fixing the latest values first, and keep walking back in history until all the required re-processing is complete. Users tend to work mostly with the latest data, so it is prudent to fix that first.

### B. Replacing Sensors

Not all data problems are software related, some involve replacing hardware. Given enough time, nearly all machines malfunction. Once a particular sensor is giving faulty data, it needs to be replaced. For energy companies, dealing with earth and subsurface structures, equipment is often installed in harsh physical conditions; thousands of feet below the

surface, or remote locations with extreme climate conditions. Therefore, swapping faulty equipment might not be a simple operation to carry out. Frequently, the equipment is not easily accessible. Surface sensors are relatively easy to replace, but those installed sub-surface in wells may require a rig to be sent on site, shut the well, and perform a work over. For this reason, sensor packs lately have redundant sensors installed; in case of a faulty sensor, just ignore the data it sends, until the work can be carried out, preferably with other work being done on the well, to minimize the cost of intervention.

### C. Changing Practices

Validating high volume, machine-generated data, in order to build the level of trust necessary for achieving business goals requires changing some practices, and developing new procedures. Before users make decisions based on this data for optimizing daily operations, or use the data for metrics in key performance indicators, or perform analysis, or use it for simulation modeling, or for creating production profiles, they must have confidence in the numbers they are dealing with.

Engineers who are validating data may need to be a little more meticulous. At first, it may require spending more time on data validation than they are used to, or have to do it more frequently than they were used to doing it in the past. This is difficult to sell, because people expect modernization and smart sensors to *reduce* their workload, not add more tasks to an already busy day.

Modernization is a transition. It is a commitment, and will result in a better way of doing things, but not without growing pains. For Big Data projects, anticipate changing how work is carried out, and prepare for it by defining a process for updating business practices. This makes it easier to implement improvements on how organizations function and results in new streamlined practices, moving them towards the overall objective of greater efficiency.

## VIII. CONCLUSION AND FUTURE WORK

Leveraging the power of data warehousing for helping establish data veracity for Big Data is a feasible and implementable solution. Most large organizations already have some data warehousing implemented; the important step to take is to make it part of the information architecture.

Digitizing an oilfield is not simply a matter of installing sensors to receive data. Equally important is strategy which recognizes the business goals and defines a framework on achieving them. In order to maximize a return on investment, and realize the benefits, information technology must play a role of shielding the end user from technical jargon and deliver information in a way that is easy to use, and

integrates seamlessly with traditionally collected data. Such a solution would provide a holistic strategy and encompass applications which integrate wholly into an upstream engineer's analytical processes.

There is plenty of work planned in the future after combining high frequency machine generated data with traditional structured data. There still remains the added value of combining the structured dataset now stored in a data warehouse with unstructured or semi-structured data. Upstream geoscientists in the oil and gas sector complement the structured data gathered in databases, with documents, observer logs, surface maps, subsurface maps, and a wide variety of schematics showing geological formations, subsurface contours, fluid migrations etc. A future Big Data solution envisions including all of these unstructured data items with the structured data gathered.

Large enterprises sanction Big Data projects and in doing so commit significant investment and resources to the effort. Some benefits are realized early for operational alerting of any outage or anomaly that requires immediate attention. The gain is fully realized only when this data is filtered and transformed into information which helps analyze trends to manage core assets such as hydrocarbon reservoirs, and wells over a long term, enhancing value in operational, tactical and strategic planning. For large enterprises, the road to harnessing the full potential of Big Data is long, but worth the journey.

### ACKNOWLEDGMENT

The author would like to thank Yasir Rafie for his valuable comments.

### REFERENCES

- [1] Gartner Inc. <http://www.gartner.com/it-glossary/big-data/> , retrieved on March 10, 2014.
- [2] D. Bartik, "Big Data: Dead by Definition, Alive in Practice" Information Week, <http://www.informationweek.com> , retrieved on Feb. 13, 2014.
- [3] T. Lukoianova, and V. Rubin, "Veracity Roadmap: Is Big Data Objective, Truthful and Credible?" Advances In Classification Research Online, 2014, doi:10.7152/acro.v24i1.14671.
- [4] IBM Inc. <http://www.ibm.com/developerworks/bigdata/karentest/veracity.html> , retrieved on March 10, 2014.
- [5] W. Wolfe, "Real Time Well Data Gathering and Analysis" unpublished.
- [6] R. D. Langenkamp, The Illustrated Petroleum Reference Dictionary, PennWell Publishing, Tusa, 1980, p. 28.
- [7] U. Nahdi, "Integrating Live and Conventional Data" unpublished.



# Parallel In-Memory Distance Threshold Queries on Trajectory Databases

Michael Gowanlock

Department of Information and Computer Sciences and  
NASA Astrobiology Institute  
University of Hawai'i, Honolulu, HI, U.S.A.  
Email: gowanloc@hawaii.edu

Henri Casanova and David Schanzenbach

Department of Information and Computer Sciences  
University of Hawai'i, Honolulu, HI, U.S.A.  
Email: henric@hawaii.edu, davidls@hawaii.edu

**Abstract**—Spatiotemporal databases are utilized in many applications to store the trajectories of moving objects. In this context, we focus on in-memory distance threshold queries that return all trajectories found within a distance  $d$  of a fixed or moving object over a time interval. We present performance results for a sequential query processing algorithm that uses an in-memory R-tree index, and we find that decreasing index resolution improves query response time. We then develop a simple multithreaded implementation and find that high parallel efficiency (78%-90%) can be achieved in a shared memory environment for a set of queries on a real-world dataset. Finally, we show that a GPGPU approach can achieve a speedup over 3.3 when compared to the multithreaded implementation. This speedup is obtained by abandoning the use of an index-tree altogether. This is an interesting result since index-trees have been the cornerstone of efficiently processing spatiotemporal queries.

**Keywords**—spatiotemporal databases; query parallelization.

## I. INTRODUCTION

Many applications require analyzing moving object trajectories (e.g., users with Global Positioning System devices, animals in ecological studies, stellar bodies in astrophysical simulations). Two relevant queries over moving object trajectories, which we term *distance threshold queries*, are: (i) Find all trajectories within a distance  $d$  of a given fixed point over a time interval  $[t_0, t_1]$ ; and (ii) Find all trajectories within a distance  $d$  of a given trajectory over a time interval  $[t_0, t_1]$ .

For instance, consider an astrobiology application that studies the habitability of the Milky Way [1]. The Milky Way is expected to host many rocky low-mass planets, some of which may be able to support complex life. The dangers to complex life include transient radiation events, such as supernovae, or close encounters with flyby stars. To model habitability one must quantify these events, which can be formulated as distance threshold queries: (i) Find all stars within a distance  $d$  of a supernova explosion, modeled as a fixed point, over a short time interval  $t$ ; and (ii) Find the stars, and corresponding time periods, that host a habitable planet and are within a distance  $d$  of a moving star,  $s$ , over the star's lifetime  $t_s$ . Given that a dataset of the Milky Way may contain billions of stellar trajectories, distance threshold queries must be performed efficiently.

Our objective is to design efficient distance threshold query processing algorithms, and we make the following contributions:

- 1) We propose a sequential algorithm that relies on an efficient trajectory indexing strategy.
- 2) We investigate parallelization in a multi-proc/multi-core environment and a General-Purpose computing on Graphics Processing Units (GPGPU) environment. The contrasting architectures require different algorithms and data structures to achieve good performance.
- 3) We outline performance bottlenecks and suggest methods for their resolution in a distributed memory environment.

## II. RELATED WORK

A trajectory is a set of points traversed by an object over time. Linear interpolation is used when processing queries that fall in between the points (i.e., one assumes that the points are connected by line segments). Most works on moving object databases propose sequential query processing algorithms that utilize an R-tree index [2] or variations of it (e.g., TB-trees [3], STR-trees [3], 3DR-trees [4], and SETI [5]). The R-tree indexes spatiotemporal data using hyperrectangular minimum bounding boxes (MBBs), where each line segment belonging to a trajectory is stored in one MBB at a leaf node. The non-leaf nodes contain the dimensions of the MBB that contains all MBBs in its sub-tree. Given a query MBB, an index search returns all leaf node MBBs that overlap the query MBB.

Except in [6], distance threshold queries have received little attention. The most related queries are  $k$  Nearest Neighbors ( $k$ NN) queries [7], [8], [9], [10]. In some sense a distance threshold query is a  $k$ NN query with an unknown value of  $k$ , since there is no a-priori limit to the number of query matches. Therefore,  $k$ NN query processing algorithms cannot be applied to process distance threshold queries.

## III. SEQUENTIAL IMPLEMENTATION

In this section we evaluate a sequential implementation of the TRAJDISTSEARCH distance threshold query processing algorithm that we proposed in [11]. Given a query trajectory line segment over a temporal extent, a query MBB is computed based on the query threshold distance. TRAJDISTSEARCH then searches a trajectory index for MBBs that overlap the query MBB. TRAJDISTSEARCH is implemented in C++, re-using an R-tree index implementation based on that initially developed by A. Guttman [2] with source code available at [12].

We run TRAJDISTSEARCH on one core of a dedicated 3.46 Ghz Intel Xeon W3690 processor with 12 MB L3 cache and sufficient memory to store the entire index. We average query response time over 3 trials. The variation among the trials is negligible so that error bars in our results are not visible. We ignore the overhead of loading the R-tree from disk into memory, which can be done once before all query processing. We measure the response time of TRAJDISTSEARCH for the following datasets and queries, which are available at [13]:

- S1: A 4-D (3 spatial + 1 temporal) dataset, *Galaxy*, containing star trajectories (for the application described in Section I), with 1,000,000 trajectory segments corresponding to 2,500 trajectories of 400 timesteps each. The query consists of 100 trajectories for 100% of their temporal extent, with a variable query distance  $d$ .
- S2: Three 4-D synthetic datasets, *Random*, with trajectories generated via random walks, with  $\sim 1,000,000$  (1M),  $\sim 3,000,000$  (3M) and  $\sim 5,000,000$  (5M) line segments corresponding to 2500, 7500 and 12500 trajectories, respectively. The query consists of 100 trajectories for 100% of their temporal extent, with a fixed query distance,  $d = 15$ .

One indexing approach is to assign each trajectory segment to its own MBB, minimizing index overlap, but maximizing the number of entries in the index. By assigning multiple trajectory segments to an MBB, index traversal time is decreased as the index contains fewer entries. However, a larger number of candidate segments is returned, many of which may not overlap the query MBB, leading to higher segment processing times. To explore the effect of varying the index resolution, for each trajectory we place its first (temporally)  $r$  segments in an MBB, its next  $r$  segments in another MBB, and so on.  $r = 1$  corresponds to using a single MBB per trajectory segment. Figure 1 plots response time vs.  $r$  for S1 and S2. A small  $r$  value can lead to high response times. In Figure 1 (a) with a value of  $r = 12$ , the response time with  $d = 5$  is 31.6 s in comparison to 186.5 s for  $r = 1$ , or a factor of 5.9 faster. Grouping multiple line segments into MBBs in this manner ensures that line segments of a trajectory are temporally contiguous. We also attempted to split trajectories so as to minimize MBB volumes using the *MergeSplit* algorithm [14], but saw no performance improvement (see full details in [11]).

#### IV. SHARED-MEMORY PARALLEL IMPLEMENTATION

##### A. Multi-core OpenMP

TRAJDISTSEARCH can be easily parallelized using OpenMP in a shared-memory setting because iterations of its outer loop are independent. Figure 2 shows the response time on the 6-core platform described in the previous section vs. the number of threads for S1 and S2 with  $r = 12$  and  $r = 10$ , respectively (i.e., the “best”  $r$  values for the sequential implementation). We see high parallel efficiency (78%-90%), with parallel speedup between 4.69 and 5.44 with 6 threads for query distances ranging from  $d = 1$  to  $d = 5$ .

##### B. GPGPU with OpenCL

In TRAJDISTSEARCH, the R-tree is used to reduce the number of line segments that must be processed. Unfortunately, the index-tree traversal is memory-bound with non-

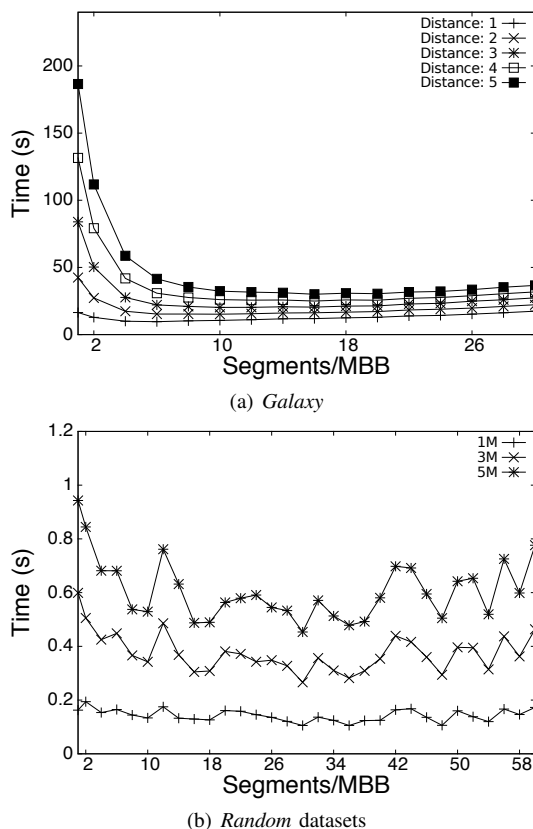


Figure 1. Response time vs. the  $r$  for (a) S1 for the *Galaxy* dataset for various query distances; and (b) S2 for the *Random-1M*, *3M*, and *5M* datasets and a query distance of 15.

deterministic execution paths due to branches, making efficient execution on a GPU challenging. As a result, we completely forego the use of the index-tree entirely so as to exploit the massive parallelism of the GPU. In the GPU version of TRAJDISTSEARCH, all line segments of each trajectory in the dataset are stored in the GPU’s global memory once and for all before all query processing. These line segments are sorted temporally based on their start times (line segments of a trajectory may not be stored contiguously). On the host, for a specified number of bins,  $B$ , we bin these line segments, where each bin is defined by a range of start times and consists of the indices of the first and last segments in the bin.

1) *Constant Sized Query Batches*: We develop a GPU kernel that processes a set  $Q$  of  $N$  query line segments, initially stored on the host. The host first sorts the line segments in  $Q$  by their start times, and determines the relevant contiguous bins that contain entry line segments that may overlap temporally with at least one query line segment. Let  $E$  denote the entry line segment index range corresponding to this set of bins. Larger values of  $B$  (i.e., smaller bins) mean a more expensive computation for  $E$  but a more precise value for  $E$  (i.e., a smaller range). Our GPU kernel takes as input  $E$  and  $Q$ , where each GPU thread is responsible for one line segment in  $E$ , and computes whether any of the segments in  $Q$  are within distance  $d$  of that line segment. This brute-force search returns relevant time intervals annotated with the IDs of the trajectories that are within the query distance. This kernel can be invoked multiple times to overlap communication and computation.

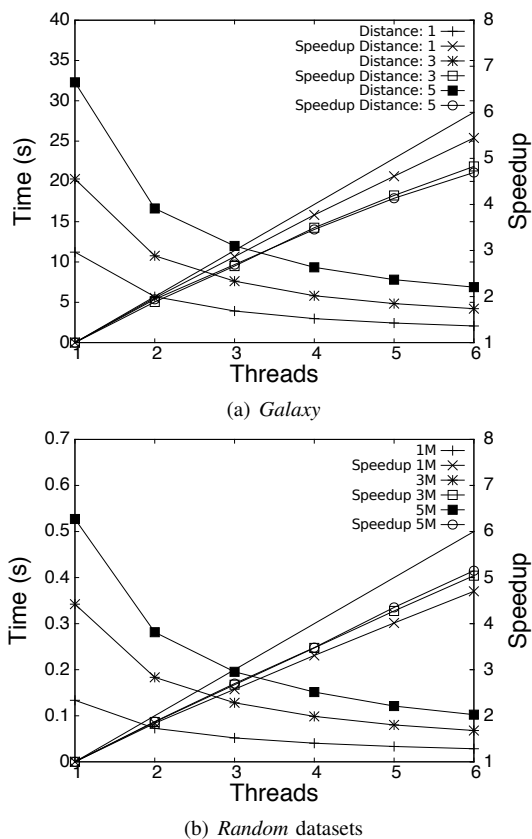


Figure 2. Response time vs. number of threads (a) S1 for the *Galaxy* dataset for various query distances and  $r = 12$ ; and (b) S2 for the *Random-1M*, *3M*, and *5M* datasets, with a query distance of 15 and  $r = 10$ .

We implemented the above kernel in OpenCL and executed it on the platform described in Section III equipped with an Nvidia Tesla C2075 GPU device. Figure 3 (a) plots response time vs.  $B$  for dataset/query S1 and for a query distance  $d = 5$ , and for various values of  $N$ . For the results in this figure we only consider one workqueue with a single kernel (1 CPU thread), so that there is no overlap of computation and communication. We find that a value of  $N = 125$  leads to the best performance, independently of the number of bins. We also see that too small a number of bins leads to high response time because the index range  $E$  is unnecessarily large. The response time plateaus around  $B = 5000$ . Figure 3 (b) shows similar results but with 3 workqueues each running an instance of the kernel, thus allowing overlap of computation and communication (using more workqueues leads to no further improvements in our experiments). We see the same trends in terms of the number of bins  $B$ . Using  $N = 100$  or  $N = 125$  leads to the lowest response time overall. The performance gain due to overlap is significant. For instance, with  $B = 5000$  and  $N = 125$ , the response time in Figure 3(a) is 2.75s while that in Figure 3(b) is 2.07s, or 24.7% faster.

In comparison to the initial sequential implementation, still for dataset/query S1, using  $r = 1$  (Figure 1 (a)) for  $d = 5$ , the GPU implementation using a single kernel (Figure 3 (a)) gives a speedup of over 67. Considering the best value of  $r = 12$  for S1, and the multithreaded CPU implementation with 6 threads, we obtain a speedup using the GPU of over 2.5 with a single kernel, and a speedup of over 3.3 when using 3 workqueues.

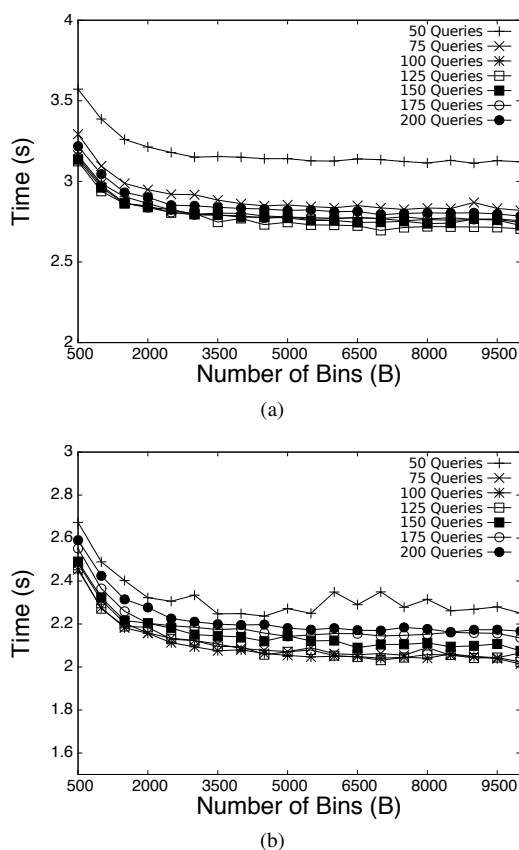


Figure 3. Response time vs.  $B$  for S1 with various  $N$  values and  $d = 5$ ; (a) 1 work queue and kernel instance; (b) 3 work queues and kernel instances.

2) *Variable Sized Query Batches*: One drawback of constant sized query batches is that, due to temporal properties of the dataset, the batch may temporally overlap with a large entry index range  $E$  but each individual query may overlap only a small subset of  $E$ , thus leading to wasteful computations. We propose here an approach that uses variable sized batches to reduce the number of these wasteful calculations. We group queries according to their temporal properties by sorting them temporally, as in the previous approach, but then binning them in the same manner as the entries described in Section IV-B. More precisely, each query line segment is assigned to one of  $C$  query bins. Each of the  $C$  bins is mapped onto the indices of the  $B$  entry bins for which the  $C$  bins overlap temporally. We construct query batches as contiguous sets of  $S$  query bins (batches contain different numbers of queries), which are sent to the GPU for each kernel invocation.

Figure 4 (a) plots response time vs.  $S$  for dataset/query S1 with  $d = 5$ . To compare with Figure 3 (a), we have plotted the average number of queries per kernel execution in Figure 4 (b). We observe that this approach performs slightly worse than the constant sized query batch approach (Figure 3 (a)). We attribute this to two factors: (i) the additional overhead required to construct the  $C$  query bins, and (ii) the fact that this particular dataset has roughly the same number of active trajectory segments at any given time. Elaborating on (ii), if the dataset contained short punctuated time periods where there are many relevant trajectories, and other periods with few relevant trajectories, then the variable sized query

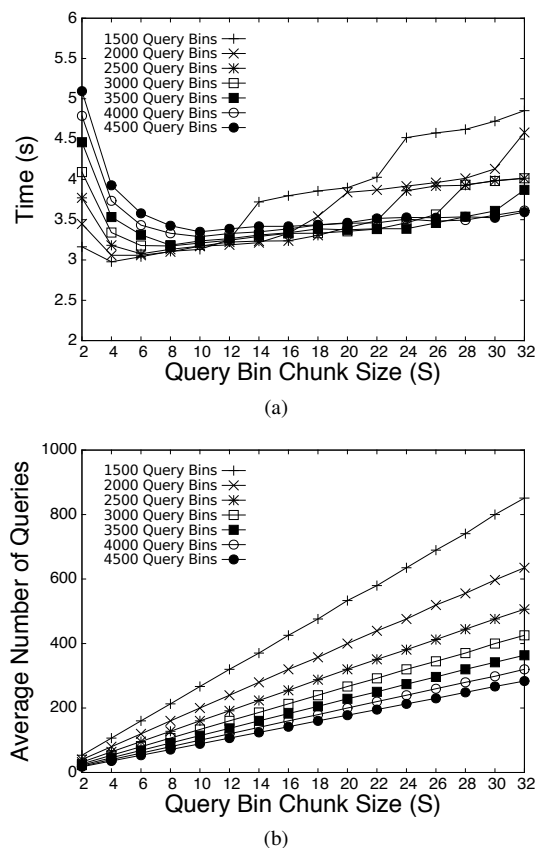


Figure 4. (a) Response time vs.  $S$  for  $S1$  with various  $C$  values,  $d = 5$ ,  $B = 7500$ , with 1 work queue and kernel instance; (b) the average number of queries per kernel execution vs.  $S$  for the results in the upper panel.

batch approach would likely outperform the constant sized query batch approach, since on average each kernel instance would be given a smaller  $E$  range. One such application would be the trajectories of vehicles, where the number of active trajectories at a given time, unlike in the case of stars in the galaxy, is influenced by human activities, such as rush hour, daytime, nighttime, etc.

## V. DISCUSSION AND CONCLUSIONS

This work in progress studies in-memory distance threshold queries for moving object trajectory databases. For a sequential implementation, a natural and effective approach is to use an index tree and to group multiple trajectory line segments into MBBs. We have shown that the resulting algorithm can be parallelized efficiently on a multi-core platform. A future work direction is to consider distributed memory environments with multiple multi-core nodes. The global index could be partitioned across the nodes, with however a high risk of load imbalance that would lead to deeper tree traversals for some of the nodes. In this context, it would be interesting to study the impact of index resolution on load balancing, possibly leading to a solution that uses different index resolutions on different nodes.

We have shown that GPGPU can lead to efficient distance threshold query processing provided queries are processed in batches of appropriate size and the use of the R-tree index is abandoned. For the queries/datasets used in our experimental

evaluation we have found that the GPU can provide a speedup of over 3.3 when compared to a multi-threaded R-tree based implementation that uses 6 cores. An important result is that our brute-force GPGPU approach outperforms the traditional search-and-refine strategy that uses the popular R-tree index. Future work in this direction may investigate the use of a GPU implementation of the R-tree [15], and possibly using hybrid approaches for splitting up the execution of the query processing algorithm between the host and the GPU device.

## ACKNOWLEDGMENTS

This material is based upon work supported by the National Aeronautics and Space Administration through the NASA Astrobiology Institute under Cooperative Agreement No. NNA08DA77A issued through the Office of Space Science.

## REFERENCES

- [1] M. G. Gowanlock, D. R. Patton, and S. M. McConnell, "A Model of Habitability Within the Milky Way Galaxy," *Astrobiology*, vol. 11, 2011, pp. 855–873.
- [2] A. Guttman, "R-trees: a dynamic index structure for spatial searching," in *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, 1984, pp. 47–57.
- [3] D. Pfoser, C. S. Jensen, and Y. Theodoridis, "Novel Approaches in Query Proc. for Moving Object Trajectories," in *Proc. of the 26th Intl. Conf. on Very Large Data Bases*, 2000, pp. 395–406.
- [4] Y. Theodoridis, M. Vazirgiannis, and T. Sellis, "Spatio-Temporal Indexing for Large Multimedia Applications," in *Proc. of the Intl. Conf. on Multimedia Computing and Systems*, 1996, pp. 441–448.
- [5] V. P. Chakka, A. Everspaugh, and J. M. Patel, "Indexing large trajectory data sets with seti," in *Proc. of the Conf. on Innovative Data Sys. Research*, 2003, pp. 164–175.
- [6] S. Arumugam and C. Jermaine, "Closest-point-of-approach join for moving object histories," in *Proc. of the 22nd Intl. Conf. on Data Engineering*, 2006, pp. 86–95.
- [7] E. Frentzos, K. Gratsias, N. Pelekis, and Y. Theodoridis, "Nearest neighbor search on moving object trajectories," in *Proc. of the 9th Intl. Conf. on Advances in Spatial and Temporal Databases*, 2005, pp. 328–345.
- [8] E. Frentzos, K. Gratsias, N. Pelekis, and Y. Theodoridis, "Algorithms for Nearest Neighbor Search on Moving Object Trajectories," *Geoinformatica*, vol. 11, no. 2, 2007, pp. 159–193.
- [9] Y.-J. Gao, C. Li, G.-C. Chen, L. Chen, X.-T. Jiang, and C. Chen, "Efficient k-nearest-neighbor search algorithms for historical moving object trajectories," *J. Comput. Sci. Technol.*, vol. 22, no. 2, 2007, pp. 232–244.
- [10] R. H. Güting, T. Behr, and J. Xu, "Efficient k-nearest neighbor search on moving object trajectories," *The VLDB Journal*, vol. 19, no. 5, 2010, pp. 687–714.
- [11] M. Gowanlock and H. Casanova, "In-Memory Distance Threshold Queries on Moving Object Trajectories," in *Proc. of the Sixth Intl. Conf. on Advances in Databases, Knowledge, and Data Applications*, 2014.
- [12] <http://www.superliminal.com/sources/sources.htm>, accessed 5-February-2014.
- [13] <http://navet.ics.hawaii.edu/%7Emike/datasets/DBKDA2014/datasets.zip>, accessed 12-February-2014.
- [14] M. Hadjieleftheriou, G. Kollios, V. J. Tsotras, and D. Gunopulos, "Efficient indexing of spatiotemporal objects," in *Proceedings of the 8th International Conference on Extending Database Technology: Advances in Database Technology*, 2002, pp. 251–268.
- [15] L. Luo, M. D. F. Wong, and L. Leong, "Parallel implementation of R-trees on the GPU," in *ASP-DAC*, 2012, pp. 353–358.

# A Database Synchronization Approach for 3D Simulation Systems

Martin Hoppen, Juergen Rossmann

Institute for Man-Machine Interaction  
RWTH Aachen University  
Aachen, Germany

Email: {hoppen, rossmann}@mmi.rwth-aachen.de

**Abstract**—Equipping a three-dimensional (3D) simulation system with database technology provides many advantages: Simulation models can be managed more efficiently than with files, temporal databases can be used to log simulation runs, and active databases provide a means for communication. Thus, we use a central database to share simulation models from different fields of application from space missions to forestry. To enable real-time access, each simulation client caches the model to its local runtime simulation database. For that purpose, each pair of databases must be synchronized. After a synchronization on schema level, each client replicates data on-demand. In this publication, we present an approach that uses both databases' notification services to keep master copies in sync with their replicate copies. State machines are used to model the approach.

**Keywords**—Database Synchronization; 3D Simulation; Distributed Database.

## I. INTRODUCTION

Simulation applications in general and 3D simulation applications in particular all follow the basic principle of applying simulation techniques to a corresponding model. Hence, the field is called modeling and simulation. A simulation model however needs some kind of data management. Up to now, files are still common for this task. In [1], we present a database-driven approach to overcome the associated disadvantages. Here, a central database is used to manage the shared simulation model, while simulation clients perform an on-demand replication of the model to their respective local, real-time capable runtime database. The central database is even used as a communication hub to drive and log distributed 3D simulations.

In this paper, we add a detailed description of the notification-based synchronization approach used in this scenario. Its specification however should be preferably universal to allow for its adoption with different database systems. For that purpose, general requirements towards the two involved database systems – generically referred to as ExtDB (the central database) and SimDB (the runtime simulation database) – were compiled [2]. They incorporate methods adopted from Model-Driven Engineering (MDE) [3] and allow to use the concepts of the Unified Modeling Language (UML) to give generalized method specifications for the different components of the overall approach [4]. Thus in this paper, the synchronization approach will also be presented using UML metaclasses.

The synchronization approach relies on change notifications. Hence, ExtDB and SimDB need an according service.

Using the notifications, the state of synchronization between both databases is monitored and modeled in a state machine for each pair of master and replicate copy. For resynchronization, transactions are scheduled and either executed or canceled out. Furthermore, notifications are used to confirm transactions and to detect change conflicts. A particular challenge in this scenario is to keep the state machine models stable, i.e., not to miss or misinterpret notifications.

The rest of this paper is organized as follows: In Section II, the foundations of the database-driven approach for 3D simulation are recapitulated. Section III summarizes the system requirements and the applied approach for method specifications using the UML metamodel. Both sections pave the way for the main Section IV where we present the notification-based synchronization approach. In Section V, exemplary applications are shown and Section VI presents some work related to our own. Finally, in Section VII, we conclude our work and present some future work.

## II. DATABASE-DRIVEN 3D SIMULATION

Using a central database (ExtDB) to manage a shared simulation model has several advantages. In contrast to a classical file based approach, databases provide a very efficient data management, well-defined access points, e.g., using a query language or an Application Programming Interface (API), a consistent data schema for structured data, and concurrent access for multiple users. This allows to persist the current state of a 3D simulation model comprising its static (e.g., building, tree, work cell) as well as dynamic (e.g., vehicle, robot) parts. During a simulation run, the state of its model's dynamic parts changes. This is an inherent property of simulation. To capture this process over time, a temporal database [5] can be used. Here, any change to the simulation model causes the previous state's conservation as a version. Altogether, this also allows to persist the course of the simulation itself. Besides these more or less passive activities, a database can also be used as an active part of the simulation. One approach is to use it as an active communication hub. An active database [5] is needed that can provide the necessary change notifications to inform clients of changes to the shared simulation model.

However, a steady, direct data exchange with ExtDB is not advisable for 3D simulation. This would lack real-time capabilities and impose a strong coupling on each and every component of the simulation system with the utilized database system. Instead, we use an approach that combines ExtDB with a local runtime database (SimDB) for each simulation client.

The lower part of Figure 1 shows the principle structure of this approach for a single pair of ExtDB and SimDB instance. By replicating required contents from ExtDB to SimDB, the simulation system can use the cached copies and the nature of ExtDB can be hidden away.

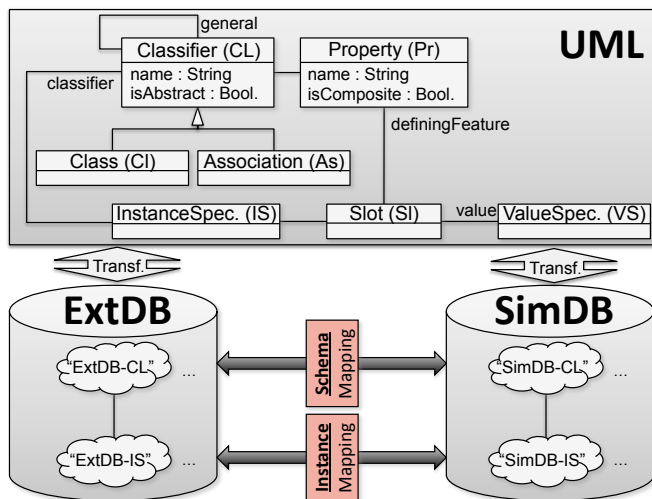


Figure 1. Principle structure of the approach for database-driven 3D simulation.

The two databases are synchronized on schema and data level. During the former, the schema description is transferred from ExtDB to SimDB so both systems "speak the same language." This builds up a schema mapping between the databases and is done once during system startup. Note however that this does not imply a semantic mapping like mapping an address represented by a single string to a fielded address representation (name, street, etc.). Instead, only the different modeling concepts (i.e., the utilized metaclasses) are mapped.

During runtime, data is loaded, i.e., replicated, from ExtDB to SimDB. Here, based on the schema mapping, the appropriate schema components are instantiated, values are copied, and an instance mapping is stored to keep the relationship between master and replicate copy. Copies no longer required can also be unloaded, i.e., removed from SimDB provided they have not been changed. Changes are tracked and resynchronized to keep both master and replicate in sync. This is realized using notification services of ExtDB and SimDB. The approach is presented in detail in Section IV.

### III. SYSTEM REQUIREMENTS

To generalize the approach system requirements were identified [2]. The aim is to make it universally available for different implementations of ExtDB and SimDB. A general compatibility of the two databases' modeling concepts is stipulated using both their metamodels. A database's metamodel represents its abstract syntax (modeling concepts). Their compatibility can then be expressed with a model transformation, e.g., using the ATL Transformation Language (ATL) [6].

To provide a common basis for arbitrary database metamodels, a pivotal metamodel with transformations from and to both databases' metamodels is stipulated as well. The pivot's metaclasses can be used to indirectly refer to SimDB's or

ExtDB's metaclasses using the demanded mapping. In the context of 3D simulation, Geographic Information Systems (GIS), Computer-Aided Design (CAD), or other 3D software, an object-oriented modeling is advisable, as such data usually consists of a huge number of hierarchically structured parts with interdependencies [5]. Thus, the UML (language unit classes) is a reasonable choice for a pivot. Figure 1 gives an overview. It also comprises the mainly utilized UML metaclasses. Altogether, this allows to generically refer to the structure of SimDB and ExtDB using UML concepts. Therefore, the method specification in the next section uses concepts like object, link, class, or property although including any database metamodel that can be mapped to the UML metamodel. Note, however, that this mapping to UML structures is conceptually needed to show the databases' compatibility and to obtain a means for generalized method specifications. The actual implementation of the synchronization approach is done on API or query language level – in particular to ensure real-time capabilities.

### IV. NOTIFICATION-BASED DATABASE SYNCHRONIZATION

Following the definition in [5], the presented scenario, i.e., the combination of SimDB and ExtDB, would be a distributed database (DDB). Similar to a distributed database management system (DDBMS), our approach aims at transparency of the distribution. However, it is a special case in which SimDB is a cache for ExtDB. Simulation clients access the shared simulation model only via SimDB. The nature and (for the most part) the existence of ExtDB are hidden away. The master copy of the simulation model is stored in ExtDB. In contrast, a classical DDB is accessed as a whole from the outside and the DDBMS hides away its distributive nature. Important DDB concepts are fragmentation, allocation and replication, as well as autonomy and heterogeneity. We use horizontal fragmentation splitting up object sets (but not objects themselves) between the central ExtDB and the connected SimDBs. All fragments are allocated to ExtDB. Further allocation, i.e., replication, to the different SimDBs is realized on-demand as shown in [4]. While ExtDB is fully autonomous SimDB is limited to the schema adopted from ExtDB. As both databases usually are different systems – e.g., SimDB is a runtime database – the assumed DDB is heterogeneous.

One or more instances of SimDB have a star-shaped connection to one instance of ExtDB. Changes are synchronized independently between each pair of SimDB and ExtDB. Differences in between such a pair are resynchronized periodically but not synchronously. Thus, we have a similar scenario as described in [7] for replication servers with asynchronous replication. However, in contrast to mobile databases, the connection is always kept alive and resynchronization is typically short-term. Furthermore, there is no global transaction or recovery manager. Changes to ExtDB by any client or to SimDB by any client component are committed without control of the synchronization component, which can merely monitor such changes. Thus, following durability (as in Atomicity, Consistency, Isolation, Durability (ACID)) they cannot be undone. Durability is important as an online (i.e., live) 3D simulation cannot be reset in the middle of a run.

One way to treat concurrent changes is an active concurrency control using locks. For distributed concurrency control,

one approach is to choose a so called distinguished copy which holds a representative lock for all its replicate copies [5]. In our case, the master copies in ExtDB could be adopted for this purpose as they are shared among all clients. However, locking is not recommendable here as acquiring locks would be time-consuming (as an ExtDB access would be necessary each time) and possible deadlocks may interrupt a running simulation.

We therefore developed a lock-free approach using notifications. For each pair of SimDB and ExtDB, the mechanism monitors changes by listening to the notifications. For resynchronization, it schedules transactions of the respective database. Due to the monitoring approach, they can only comprise a single data operation. The approach is similar to optimistic concurrency control (OCC) [7]. However, transactions cannot be rolled back when changes are conflicting. Instead, conflicts are only implicitly resolved: The last client changing a value is given precedence. Altogether, it is crucial that the synchronization component always knows about the state of synchronization for each copy. However, besides resynchronization and passive monitoring, the mechanism cannot and must not intervene, e.g., by rejecting changes as mentioned above.

### A. Change Tracking

For each pair of SimDB and ExtDB, a change tracking component connects to the notification services of SimDB for so-called *internal* notifications and of ExtDB for so-called *external* notifications. Notifications include insertions and removals of objects and links, as well as updates of object properties. A link between objects can only be removed or inserted but not updated, as its identity is only derived from the connected objects (and the corresponding association on schema level).

For the sake of simplicity, external notifications from ExtDB are abbreviated as extInsert, extUpdate, and extRemove, internal notifications from SimDB as simInsert, simUpdate, and simRemove, accordingly. During runtime, these notifications are evaluated. Depending on the current state of the corresponding pair of master and replicate copy represented by an instance mapping entry, a transaction may be scheduled that can later be used to resynchronize the detected change from the one to the other database. A scheduled transaction comprises one data operation with its kind (insert, remove, or update), the affected instance (object or link) or its id, and for updates the affected property. A transaction for transferring a change from SimDB to ExtDB is called an *out-bound* transaction and will be abbreviated with the prefix *sim2ext*. For example, when detecting an object insertion within SimDB by a simInsert notification, a new sim2extInsert out-bound transaction may be scheduled. Its (future) execution will insert an equivalent object of the corresponding ExtDB-Classifer (using the schema mapping) into ExtDB. Here, the current property values are retrieved from the SimDB object's slots and are replicated for the new ExtDB object. Finally, the new object complements the corresponding instance mapping entry with its identifier. This can be seen as the complementing operation to the loading of objects. Links are treated accordingly but without the need for property value replication. An instance's removal (object or link) from SimDB, notified

by a simRemove notification, may lead to a sim2extRemove transaction whose (future) execution will remove the associated ExtDB instance. A simUpdate notification signals the change of a SimDB object's property and may be scheduled as a sim2extUpdate transaction to transmit the value change from SimDB to ExtDB. Similar to sim2extInsert transactions, a sim2extUpdate transaction's execution retrieves the current value of its corresponding property from SimDB and replicates it to ExtDB.

Accordingly, external notifications may lead to the scheduling of *in-bound* transactions for resynchronizing global changes from ExtDB to SimDB. They are prefixed by *ext2sim*: ext2simInsert, ext2simRemove, and ext2simUpdate. Responses to external notifications are mostly identical to their internal counterparts. However, due to the nature of SimDB being a cache for ExtDB, a variation applies when treating external insertions. New objects or links within ExtDB may be handled by different strategies. They may be ignored or subsequently taken into account by a loading transaction (ext2simInsert). In this paper, the latter approach is chosen. Alternatively, one could consider to reevaluate previously executed queries to determine the "interest" in the new instance.

Altogether, instance mapping entries (i.e., pairs of master and replicate copy) can be seen as to reside in a certain state of synchronization. This can be modeled as a state machine in statechart notation [8] for each object's or link's instance mapping entry. For objects, this state machine is given in Figure 2 (it is similar for links). It may be in a synchronous state (*Synced*), a *Loading* or *Unloading* state, a state representing its absence or non-management (*NonManaged*), or a transaction state (*ext2simInsertPending*, *ext2simRemovePending*, etc.). For update transactions, the synchronization states of an object's properties are concurrently modeled in the sub states of state UpdatesPending shown in Figure 5.

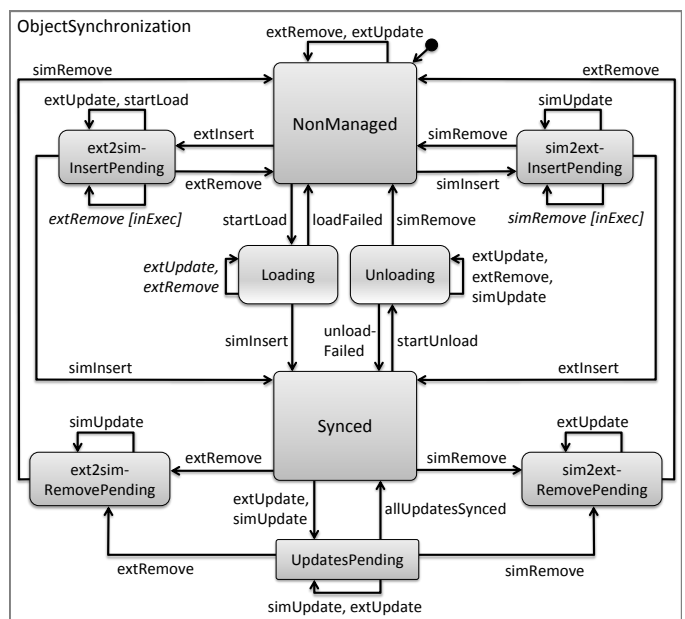


Figure 2. Synchronization states of an object's instance mapping.

An exemplary chain of events depicted in Figure 3 would be the insertion of a new door object into SimDB leading

to a transition guarded by `simInsert` from the initial state `NonManaged` to state `sim2extInsertPending` shown in Figure 4. Here, a `sim2extInsert` transaction is scheduled for the new object. When the transaction is executed (see Subsection IV-C), an equivalent object is inserted into `ExtDB` eventually causing the database to issue an `extInsert` notification. In turn, this event triggers a transition from the `sim2extInsertPending` to the `Synced` state. Thus, the `extInsert` event confirms the insertion into `ExtDB` and is used as a receipt to acknowledge a transaction’s successful execution. This is especially useful for handling concurrent changes within `SimDB` and `ExtDB` occurring during other transaction’s execution.

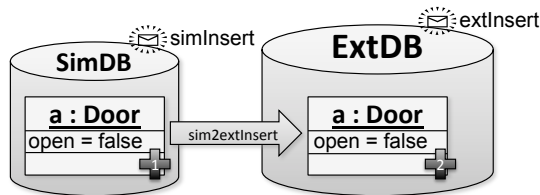


Figure 3. Exemplary insertion of a door object into `SimDB` and subsequent synchronization to `ExtDB` using a `sim2extInsert` transaction.

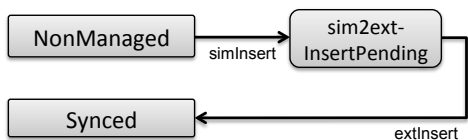


Figure 4. Excerpt from Figure 2 for the state transitions accompanying the exemplary insertion depicted in Figure 3.

The receipt handling mechanism is also used to handle mutual changes that cancel each other out. An example are mutual removals: An instance is, e.g., first removed from `ExtDB` and subsequently from `SimDB` by independent processes. Thus, a previously scheduled `ext2simRemove` transaction with pending execution (in state `ext2simRemovePending`) is canceled out by the incoming `simRemove` notification for the same instance. The event causes a transition to the `NonManaged` state.

Property changes are modeled in Figure 5. The `UpdatesPending` state encapsulates a sub state structure for managing property updates. Primarily, it contains a super state `UpdatesPr` with concurrent regions for each of the object’s properties, e.g., region `UpdatesPri` for the object’s  $i^{th}$  property. A region for Property  $Pr_i$  has three states representing an unchanged property value (`SyncedPri`), a property value changed within `SimDB` (`sim2extUpdatePendingPri`), and a property value changed within `ExtDB` (`ext2simUpdatePendingPri`). Further updates to the object’s value for  $Pr_i$  can be ignored when they stem from the same database (i.e., both `SimDB` or both `ExtDB`), as the new value has to be transferred to `SimDB`, anyway. However, a subsequent update to the same property from within `SimDB` causes a change conflict (see Subsection IV-B). The modeled strategy is to give precedence to the more recently notified change. Thus, a transition to `sim2extUpdatePendingPri` is triggered. When the transaction implicitly scheduled on entering one of the update states is executed, a notification is needed as a receipt. However, in contrast to insert or remove transactions, there is no “natural” counterpart for update transactions. An executed

`ext2simUpdate` transaction causes a `simUpdate` notification that is indistinguishable from any other third party changes. Thus, before execution, an “inExec” flag is set. For `ext2simUpdate`, the next `simUpdate` notification for  $Pr_i$  will trigger a transition back to the synced state of this property (the `inExec` flag will be reset). When all concurrent regions are in their respective synced state, a synchronized (in terms of concurrency) transition to the `Done` state is triggered (modeled by the vertical bar). On entering this state, the `allUpdatesSynced` event is raised triggering a transition from the super state `UpdatesPending` to the `Synced` state (see Figure 2).

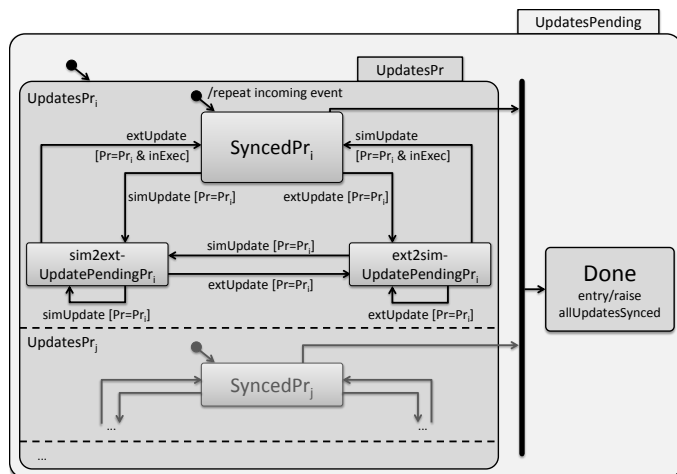


Figure 5. Sub structure of state `UpdatesPending` from Figure 2 for property updates.

In some situations, events may also be ignored. Within the state machines, this may be modeled as self-transitions. For example, in `sim2extRemovePending`, further `extUpdate` events from `ExtDB` can be ignored as the corresponding object will be removed from `ExtDB`, anyway.

### B. Change Conflict Handling

As mentioned above, changes (insertions, removals, and updates) from `ExtDB` and `SimDB` may conflict when they occur to the same instance (and property) before executing the corresponding transaction. For example, in a city scenario, a building’s street number is locally changed within `SimDB` causing a `sim2extUpdate` transaction. Before this change is made persistent and globally available within `ExtDB` by executing the transaction in a resynchronization run, the very same number is changed within `ExtDB` (e.g., by another simulation client). Following the strategy modeled above, the previous change is omitted and instead a new `ext2simUpdate` transaction is stored.

In general, different strategies to handle such situations could be thought of. First of all, conflicts can be avoided beforehand by giving only mutual exclusive write access to instances. This approach could be used in distributed simulation scenarios where separate objects are simulated by different clients without interaction. This can be managed by a superordinate simulation control. Avoiding the occurrence of conflicts could also be realized by explicitly locking changed instances or their property value in the respective other database. How-



ever, this may stall or even reset a simulation run as mentioned above.

Thus, a monitoring, i.e., reactive handling of change conflicts as mentioned above is inevitable. The presented methods' strategy is embedded in the given state machines. For change conflicts, two scenarios can be distinguished: A conflict may either occur *before* or *during* a transaction's execution. Before executing a transaction, conflict handling can be realized straightforward. It is modeled with simple transitions within the state machines. One example is the precedence for more recently notified updates as shown above. Another strategy is that object removals are final and thus "always win". Id est, a pending remove transaction for an object precedes all update events for the object. For pending object insertions, conflicts cannot occur as the corresponding object does not exist in the respective other database.

As long as a transaction is still pending, incoming events can always be processed by state transitions to reflect the relation between SimDB and ExtDB. In a resynchronization run, the current state of each state machine is evaluated (compare Subsection IV-C). If a state with pending transaction  $T1$  is determined  $T1$  is executed. However, this decision is made independently at each client. A notification from a previously committed, conflicting transaction  $T2$  may arrive just after  $T1$ 's execution is started. In some cases,  $T1$  may still be abortable. But the notification may just as well arrive when  $T1$  commits. So, while native transactions of the utilized database management systems (DBMSs) themselves are usually isolated the decision to start a pending transaction is not. This limits transaction isolation (i.e., ACID properties) in the distributed system.

The same applies to the reading of property values. Objects can be removed, and links can be removed and inserted based only on the information from the corresponding notification. For object insertions and property updates however, the current state of the respective source database has to be retrieved as notifications themselves do not contain the corresponding values. Thus, when such a transaction is executed the source values may have already been changed by subsequent transactions whose notifications may either have not yet arrived or transaction execution may already have started as described above. This also limits transaction isolation.

Thus, a strategy had to be found for dealing with such situations. Otherwise, scenarios where a change in one database is neither reflected within the other database nor within the instance mapping's state machine may occur. For example, a property value is changed in ExtDB, but its instance mapping's state machine is in state Synced although SimDB still holds the previous value.

The primary instrument to handle such interfering changes is the aforementioned usage of notifications as receipts. For that purpose they must have the following features:

- 1) A notification's arrival guarantees the corresponding operation to be executed.
- 2) The order of arrival of a single database's notifications is identical to the execution order of the corresponding operations.
- 3) Between one running instance of SimDB and ExtDB

there is at most one transaction being executed at a time (see Subsection IV-C).

Based only on these assumptions, a conflict management can be stable. However, one should keep in mind:

- 1) A notification not yet received does not imply that the corresponding operation is not yet executed (notifications may be delayed).
- 2) On arrival of a notification, the *current* state within the database must not be consulted for further state transitions. By time of arrival it may already have been changed several times.
- 3) The order of arrival between notifications from ExtDB and notifications from SimDB is arbitrary.

Based on these considerations, a special event handling can be implemented to process the queued events after a transaction's execution. As stated above, the main problem are notifications arriving between the start of a transaction's execution and the arrival of the corresponding receipt notification. For a proper event handling, these events must sometimes be reordered. To be precise, they are captured and reinserted into the event queue just after the receipt event. This ensures their correct processing in terms of state transitions. The procedure is necessary for object or link insertions, link removals, object updates, and object or link loading. In the state machines, transitions with italic text particularly model this case. In the sub states of UpdatesPending, this highlighting is omitted as the same transitions are needed for standard and for this special event handling.

One example are updates (Figure 5). A property's update transaction can be examined separately as updates of different properties are independent from each other. Table I lists an exemplary sequence of events for some integer property and the associated actions, statemachine states, values in SimDB and ExtDB, and emitted notifications. In the example, the local property's slot value in SimDB is updated several times even while changes are replicated to ExtDB. Notifications are used to ensure that all updates are reflected within the statemachine's current state.

Initially (step #1), SimDB and ExtDB are in sync at value 10. The value in SimDB is changed to 20 (#2) and the corresponding simUpdate notification (a) triggers a statemachine transition (#3). At some point in time, the client starts the resynchronization process (#4). Then, a first interfering update (#5) changes the value to 30. As property update notifications do not contain a value it must be retrieved from the respective database at transaction execution time (#6). Afterwards, a second interfering update (#7) changes the value to 40. In #8, the read value 30 is replicated to ExtDB. As mentioned above, the order, in which notifications from SimDB and ExtDB are received, is arbitrary. Thus, notifications simUpdate (a) and (b) may be processed first (#9, #10). As the "inExec" flag is set, all notifications are stored (instead of ignored without the "inExec" flag being set) until the corresponding receipt notification extUpdate is processed in #11. Subsequently, the flag is reset and both stored notifications are reinserted into the event queue. While the receipt notification eventually yields a transition back to the Synced state (#12), notification reinsertion causes the necessary transition back to the state of pending updates (#13) to replicate the value of 40 from

TABLE I. EXAMPLE OF A LOCAL INTERFERING UPDATE OF SOME INTEGER PROPERTY WITHIN A SINGLE SIMDB.

#	action	statemachine	SimDB val.	ExtDB val.	notification
1	(initial state)	Synced	10	10	
2	update 10 → 20 in SimDB		20		simUpdate (a)
3	process event simUpdate (a)	→ UpdatesPending / sim2extUpdatePendingPr <sub>i</sub>			
4	start resync	inExec := true			
5	update 20 → 30 in SimDB (1st interference)		30		simUpdate (b)
6	read current value from SimDB				
7	update 30 → 40 in SimDB (2nd interference)		40		simUpdate (c)
8	execute transaction sim2extUpdate			30	extUpdate
9	process event simUpdate (b)	[inExec=true] ⇒ store simUpdate (b)			
10	process event simUpdate (c)	[inExec=true] ⇒ store simUpdate (c)			
11	process event extUpdate	→ UpdatesPending / SyncedPr <sub>i</sub> → Done			allUpdatesSynced
		inExec := false			
		reinsert simUpdate (b) and simUpdate (c) in event queue			
12	process event allUpdatesSynced	→ Synced			
13	process event simUpdate (b)	→ UpdatesPending / ext2simUpdatePendingPr <sub>i</sub>			
14	process event simUpdate (c)	(self-transition)			
15	start resync ...	...			

SimDB to ExtDB. The additional simUpdate notification (c) only yields a self-transition (#14) as an update is already pending. Another resynchronization run would replicate the value to ExtDB starting at #15.

This approach to capture and reinsert notifications is needed as it is unknown whether an interfering update was done before (#5) or after (#7) reading the current value from SimDB in #6 to execute the sim2extUpdate transaction in #8. Note that when only interfering updates of the first type occur, the additional simUpdate notifications are in fact redundant. However, this is acceptable to guarantee that no updates are lost between SimDB and ExtDB. In case of interfering updates from other clients to ExtDB, additional extUpdate (instead of simUpdate) notifications are emitted. Here, notifications need not be stored as the first extUpdate notification is simply interpreted as the expected receipt and subsequent extUpdates yield normal state transitions. Finally, the same store-and-reinsert strategy is used similarly in the other use cases mentioned above (object insertions, link removals, and object or link loading).

Altogether, as mentioned above, this approach cannot avoid or fix conflicts but only detect them and react on them. However, the utilized SimDB and ExtDB themselves are not corrupted as they provide safe standard database access methods. Thus, only the distributed synchronization state must be kept free of corruptions. This is ensured by the presented approach.

### C. Resynchronization

In resynchronization, all scheduled transactions are executed to bring the two databases back in sync. This process can be triggered in several ways. When the approach is applied in a collaborative scenario, it can be initiated manually. For immediate response from and to other users, it can also be automatically triggered after each transition to a state with pending transaction. In distributed simulation, typical access patterns include constantly repeated changes of the same few property values, e.g., a moving car and a moving helicopter. In such scenarios, transactions can be aggregated within short

but arbitrary periods to lower the impact on traffic. However, this includes a trade-off between traffic and update rate.

## V. APPLICATIONS

Using the presented approach, different kinds of applications have already been realized as shown in Figure 6.

In a city scenario, a central database (ExtDB) manages a shared simulation model with a city, a helicopter, and a car. Two simulation clients are connected with their respective synchronized SimDB and each control a vehicle. Changes (e.g., the movement of the car or the helicopter) are distributed using the methods presented in this paper. Furthermore, all changes are automatically archived using a temporal ExtDB. This provides an integrated log for the development of the simulation model's dynamic properties over time and allows for subsequent replay, analysis, debriefing, and archiving. Usually, such applications use amounts of files for data management combined with a decentralized communication infrastructure, e.g., based on the High Level Architecture (HLA) [9], and separate logging components are needed to archive a simulation. In contrast, we provide a more integrated approach. This avoids divergence between data management and the corresponding change distribution mechanism, no separate mechanism is needed to access logged data, and a consistent data schema provided by the central database is used throughout the distributed system.

In another scenario, a planetary landing mission is simulated. During descent, a database-managed, shared model of the planet's surface (i.e., an object-oriented map) is created by different components in a distributed approach. Subsequently, the same map can be used for (simulated) navigation. All system components benefit from using and building up the same shared model with a consistent schema, standardized interfaces, and an integrated communication infrastructure using the presented approach.

As a last example, a forest model is extracted from remote sensing data and other geo data sources. Here, the approach is used by the various stakeholders in the forest sector to collaboratively generate, update, refine, analyze, simulate, and



Figure 6. Different applications realized using the presented approach.

simply use the highly detailed forest model managed by an ExtDB component. Instead of directly accessing this central database, the presented approach decouples clients from the utilized technology of ExtDB by only accessing data from their local SimDB database. Furthermore, the very same data schema can be used throughout the applications reducing "friction losses" due to (offline) data conversions.

## VI. RELATED WORK

Regarding database synchronization for 3D simulation systems and similar software only few approaches can be found. In [10], a combination of scene-graph-based 3D clients with a federation of databases connected by the Common Object Request Broker Architecture (CORBA) is proposed. On client-side, a local object-oriented DBMS (OODBMS) provides an in-memory scene object cache connected to the federation using an Object Request Broker (ORB). Cached objects are bidirectionally replicated to the scene graph. Concurrency control among the federated databases and the local object caches allows multi user interaction between the clients.

A mobile Augmented Reality (AR) system combining distributed object management with object instantiation from databases is described in [11]. Objects are distributed shallowly by creating "ghost" copies retaining a master copy only at one site. Such a ghost is a non-fully replicated copy of its master allowing simplified object versions to be transmitted (e.g., with sufficient parameters for rendering). Changes to the master copy are pushed to all its ghosts. Remote systems can change a master copy by sending it a change request.

In [12], [13], a Virtual Reality (VR) system is combined with an OODBMS to provide VR as a multi-modal database interface. In [14], a revised version adds collaborative work support. For update propagation, VR clients issue changes to the shared virtual environment as transactions to the back-end they are connected to. After an interference check they are committed to the database and distributed by a separate notification service. The system uses transactions with regular ACID properties (e.g., for "Create box B") committed as a whole as well as special continuous transactions for object movements. For the latter, atomicity does not apply as movements are committed incrementally to frequently propagate updates.

The "Collaborative Urban Planner" described in [15] is based on the multi-user Virtual Environment system DeepMatrix [16], extended by a relational DBMS back-end providing persistency. Clients allow for so-called shared operations like "rotate object" that are sent to the server for distribution and persistency. A server application provides concurrency control, message distribution and data management. It represents the single point of access to the database ensuring consistency among the clients' shared operations. The database primarily contains meta information on shared objects (position, texture).

In [17], a "Virtual Office Environment" contains 3D data and semantics managed by a DBMS to allow semantic-based queries and collaboration. Clients' actions are issued as queries to the shared database. Changes are distributed to all other clients, which adopt them locally.

A "shared mode" for database-driven collaboration is presented in [18]. In a chess application example with two players a shared database with the game's setting is alternately updated by the one client while being polled for changes by the other, which subsequently reflects the changes in his own virtual scene instance.

Compared to our approach, [10] comes close but lacks details and is only a proposal without known implementations. The ghosts in [11] may suffice for rendering but are to restricted for sophisticated simulation applications. Furthermore, not all objects are managed by the database. In [12], [13], [14], [17], only VR-specific data and operations are supported. [15] does not manage the model data itself using the database. Finally, the approach in [18] is similar to our own but only demonstrates a very limited type of change distribution. Altogether, no other approach offers a comparably tight integration of database technology into 3D software or simulation systems.

Similarities to our MDE-based approach for the general assessment of database compatibility can be found in generic model management. [19] introduces different generic schema operations like match, merge, translate, diff, and mapping composition. The work gives an overview but concentrates on tool support for semi-automatic mappings. Our own approach can be seen as an implementation of the "ModelGen" operator that automatically translates a schema from one metamodel into another, including mapping creation. However, in contrast, we provide an automatic mapping of schemata and a runtime approach instead of a static mapping.

Another implementation is provided in [20]. A pivotal supermodel is used to transform schema as well as data. In [21], the same system is extended to provide runtime transformations with read-only access. A similar approach is taken in [22] using a proprietary pivotal graph-based representation. [23] presents an approach for transforming schema and data between the Extensible Markup Language (XML) and the Structured Query Language (SQL). However, none of these approaches use standardized metamodeling and model transformation languages as used in our approach.

## VII. CONCLUSION AND FUTURE WORK

We presented an approach for synchronizing a central database (ExtDB) with simulation databases (SimDB) as a

basis for database-driven 3D simulation. After recapitulating our previously published background of the approach, the main contribution of this work is presented in detail: The core method for synchronization. For each pair of master and replicate copy it manages the state of synchronization – modeled as a state machine. It is based on notifications provided by both databases. On the one hand they are used to track the changes and schedule transactions for subsequent resynchronization. On the other hand, they are used as receipts to acknowledge transaction execution and to detect change conflicts. Compared to other methods for collaboration in 3D software systems, this approach provides a tight integration of advantages from the database field into simulation technology. Different applications already prove its practicability.

In future, we will examine further applications, e.g., from the field of industrial automation. Moreover, a porting of the approach to other database systems than the current prototypes will be reviewed. Finally, the integration of temporal databases will be examined in further detail, especially for valid time, bitemporal, or multi-temporal databases.

#### REFERENCES

- [1] M. Hoppen, M. Schluse, J. Rossmann, and B. Weitzig, "Database-Driven Distributed 3D Simulation," in Proceedings of the 2012 Winter Simulation Conference, 2012, pp. 1–12.
- [2] M. Hoppen, M. Schluse, and J. Rossmann, "A metamodel-based approach for generalizing requirements in database-driven 3D simulation (WIP)," in Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium, ser. DEVS 13. San Diego, CA, USA: Society for Computer Simulation International, 2013, pp. 3:1–3:6.
- [3] M. Brambilla, J. Cabot, and M. Wimmer, Model-Driven Software Engineering in Practice, ser. Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers, 2012.
- [4] M. Hoppen, M. Schluse, and J. Rossmann, "Database-Driven 3D Simulation - A Method Specification Using The UML Metamodel," in 11th International Industrial Simulation Conference ISC 2013, V. Limère and E.-H. Aghezzaf, Eds., Ghent, Belgium, 2013, pp. 147–154.
- [5] R. Elmasri and S. B. Navathe, Database Systems: Models, Languages, Design, And Application Programming, 6th ed. Prentice Hall International, 2010.
- [6] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, "ATL: A model transformation tool," Science of Computer Programming, vol. 72, no. 1-2, Jun. 2008, pp. 31–39.
- [7] T. Connolly and C. Begg, Database systems: a practical approach to design, implementation, and management, internatio ed. Pearson Education (US), 2009.
- [8] D. Harel, "Statecharts: a visual formalism for complex systems," Science of Computer Programming, vol. 8, no. 3, Jun. 1987, pp. 231–274.
- [9] Simulation Interoperability Standards Committee (SISC), "Standard for Modeling and Simulation High Level Architecture (HLA) IEEE 1516," 2000.
- [10] E. V. Schweber, "SQL3D - Escape from VRML Island," 1998. [Online]. Available: <http://www.infomaniacs.com/SQL3D/SQL3D-Escape-From-VRML-Island.htm>
- [11] S. Julier, Y. Baillot, M. Lanzagorta, D. Brown, and L. Rosenblum, "Bars: Battlefield augmented reality system," in NATO Symposium on Information Processing Techniques for Military Systems, 2000, pp. 9–11.
- [12] Y. Masunaga and C. Watanabe, "Design and implementation of a multimodal user interface of the Virtual World Database system (VWDB)," in Proceedings Seventh International Conference on Database Systems for Advanced Applications. DASFAA 2001. IEEE Comput. Soc, 2001, pp. 294–301.
- [13] Y. Masunaga, C. Watanabe, A. Osugi, and K. Satoh, "A New Database Technology for Cyberspace Applications," in Nontraditional Database Systems, Y. Kambayashi, M. Kitsuregawa, A. Makinouchi, S. Uemura, K. Tanaka, and Y. Masunaga, Eds. London: Taylor & Francis, 2002, ch. 1, pp. 1–14.
- [14] C. Watanabe and Y. Masunaga, "VWDB2: A Network Virtual Reality System with a Database Function for a Shared Work Environment," in Information Systems and Databases, K. Tanaka, Ed., Tokyo, Japan, 2002, pp. 190–196.
- [15] T. Manoharan, H. Taylor, and P. Gardiner, "A collaborative analysis tool for visualisation and interaction with spatial data," in Proceedings of the seventh international conference on 3D Web technology. ACM, 2002, pp. 75–83.
- [16] G. Reitmayr, S. Carroll, A. Reitemeyer, and M. G. Wagner, "DeepMatrix - An open technology based virtual environment system," The Visual Computer, vol. 15, no. 7-8, Nov. 1999, pp. 395–412.
- [17] K. Kaku, H. Minami, T. Tomii, and H. Nasu, "Proposal of Virtual Space Browser Enables Retrieval and Action with Semantics which is Shared by Multi Users," in 21st International Conference on Data Engineering Workshops (ICDEW'05). IEEE, Apr. 2005, pp. 1259–1259.
- [18] K. Walczak and W. Cellary, "Building database applications of virtual reality with X-VRML," in Proceeding of the seventh international conference on 3D Web technology - Web3D '02. New York, New York, USA: ACM Press, Feb. 2002, pp. 111–120.
- [19] P. A. Bernstein and S. Melnik, "Model management 2.0: manipulating richer mappings," in Proceedings of the 2007 ACM SIGMOD international conference on Management of data - SIGMOD '07. New York, New York, USA: ACM Press, Jun. 2007, pp. 1–12.
- [20] P. Atzeni, P. Cappellari, and P. Bernstein, "Model-Independent Schema and Data Translation," in Advances in Database Technology - EDBT 2006, ser. Lecture Notes in Computer Science, Y. Ioannidis, M. Scholl, J. Schmidt, F. Matthes, M. Hatzopoulos, K. Boehm, A. Kemper, T. Grust, and C. Boehm, Eds. Springer Berlin / Heidelberg, 2006, vol. 3896, pp. 368–385.
- [21] P. Atzeni, L. Bellomarini, F. Bugiotti, and G. Gianforme, "A runtime approach to model-independent schema and data translation," in Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, ser. EDBT '09. New York, NY, USA: ACM, 2009, pp. 275–286.
- [22] A. Smith and P. McBrien, "A Generic Data Level Implementation of ModelGen," in Sharing Data, Information and Knowledge, ser. Lecture Notes in Computer Science, A. Gray, K. Jeffery, and J. Shao, Eds. Springer Berlin / Heidelberg, 2008, vol. 5071, pp. 63–74.
- [23] P. Berdager, A. Cunha, H. Pacheco, and J. Visser, "Coupled Schema Transformation and Data Conversion for XML and SQL," in Practical Aspects of Declarative Languages, ser. Lecture Notes in Computer Science, M. Hanus, Ed. Springer Berlin / Heidelberg, 2007, vol. 4354, pp. 290–304.

# Achieving High Availability in D-Bobox

Miroslav Cermak, Filip Zavoral  
 Faculty of Mathematics and Physics  
 Charles University in Prague, Czech Republic  
 {cermak, zavoral}@ksi.mff.cuni.cz

**Abstract**—Using a distributed environment for data stream processing brings many challenges, especially when requiring an exact result from processing of big data. A distributed system is more vulnerable to failures as hardware crashes, software errors, or network malfunctions. Loss of node current state and loss of intermediate results due to node failure results in the restart of the computation, which increases the time of the computation and its cost and this is therefore unacceptable. Achieving high availability (HA) of such system brings some challenges. In this paper, we introduce our framework for parallel and distributed processing, D-Bobox, and its requirements on high availability implementation. We also describe the main high availability methods used today and discuss their applicability in our framework. Finally, we propose a solution how to obtain high availability in D-Bobox.

**Keywords**—high availability; D-Bobox; stream computing; distributed computing;

## I. INTRODUCTION

A new class of applications - stream processing systems (SPS) - is given a lot of attention in the last years [1], [2], [3]. These applications have to process high amount of low latency data streams, i.e., financial data processing, patients monitoring using various sensors, traffic analysis, etc. Stream processing seems to be effective not only in processing continuous data streams, but also in processing big static data as for example semantic databases [4]. Distributing sources and processing of big data at multiple nodes allows better scaling of computation performance in terms of data size. Stream processing system that uses advantages of the distributed environment are called distributed SPS (DSPS).

A typical approach to increase the performance of a distributed system is adding more computational nodes. However, this also increases the risk of a failure, which has a negative effect on the performance and dependability of a distributed system. Faults introduce errors into computation so we get wrong or incomplete results. However, many applications require that the system provides exact and same results each time running on the same input data. One of such systems can be a database system that also requires performance effectiveness and good performance/value ratio. Therefore, the presence of a high availability (HA) unit that handles recovery fast and correctly with minimal impact on failure-free processing is necessary for DSPSs.

To achieve HA in distributed stream processing systems, following tasks must be addressed:

- 1) periodic and incremental backup (or replication) of computing node state
- 2) error detection

- 3) choosing a failover node
- 4) lost state recovery after failure
- 5) manage network partition

In this work, we deal with the recovery from a node failure, so we pay our attention mainly to tasks (1), (3) and (4) as they have the biggest impact on the behavior and characteristic of each of the HA methods. Since error detection is mostly independent from the actual HA (recovery) method and the management of network splitting and partitioning is a specific type of failure concerning multiple nodes, we do not address these issues in this paper.

The paper is structured as follows: in Section II, we define recovery types according to [5]. In Section III we present contemporary HA methods that are discussed on the selected HA problems. The D-Bobox system is introduced in Section IV; in Section V we propose solutions for the integration of HA into D-Bobox.

## II. RECOVERY TYPES

The ability to mask failure so it cannot be observed from final data stream is considered the fundamental requirement on HA algorithms. Consider a node  $U$ , that contains a set of  $n$  input data streams ( $I_1, \dots, I_n$ ) and produce the output stream  $O$ . Computation  $e$  consists of processes like consuming, processing and producing data tuples. The data stream  $O_e$  is the result of the computation  $e$  at the node  $U$ . According to their handling of the  $O_f + O' = O$  equality, where  $O_f$  is computation result before failure,  $O'$  is output after recovery and  $O$  is failure free computation output, recovery types can be named as *gap recovery*, *rollback recovery* and *precise recovery*.

1) *Gap recovery*: is the simplest and least demanding recovery type. It manages node replacement after node failure detection. However, input and output preservation is not guaranteed and data loss is expected. Its main advantage is fast recovery time and almost no slowdown of failure free execution.

2) *Rollback recovery*: ensures that no information is lost during failure. According to operators used, the rollback recovery can be further divided into following subtypes:

- *repeatable* when exactly the same tuples are generated during recovery.
- *convergent* when different tuples from the same data are generated, and these tuples converge to the original tuples.
- *divergent* when different tuples from the same data are generated, and these tuples never converge to the

TABLE I. OUTPUTS PRODUCED BY EACH TYPE OF RECOVERY

Recovery type	Before failure			After failure				
	$t_1$	$t_2$	$t_3$	$t_4$	$t_5$	$t_6$	$t_7$	...
Precise	$t_1$	$t_2$	$t_3$	$t_4$	$t_5$	$t_6$	$t_7$	...
Gap recovery	$t_1$	$t_2$	$t_3$			$t_6$	$t_7$	...
Rollback								
- repeatable	$t_1$	$t_2$	$t_3$	$t_2$	$t_3$	$t_4$	$t_5$	...
- convergent	$t_1$	$t_2$	$t_3$	$t'_4$	$t'_5$	$t_6$	$t_7$	...
- divergent	$t_1$	$t_2$	$t_3$	$t'_4$	$t'_5$	$t'_6$	$t'_7$	...

original ones. This is typical for non-deterministic operators.

3) *Precise recovery*: guarantees the strongest recovery by completely masking failures, so the output after failure is same as the output without failure.

Table I shows output streams of the mentioned recovery types. Each stream consists of the sequence of tuples  $t_i$  before and after the failure. As we can see, precise recovery contains a sequence that is identical to a sequence without failures. In case of gap recovery, there are some tuples missing; they create undesirable gaps in the data stream. The output of the repeatable rollback recovery contains identical tuples as in the regular output, but some of the tuples are duplicated. On the other hand, the outputs of the convergent and divergent rollback recoveries contain different tuples  $t'_j$  after failure. In case of the convergent recovery, different tuples became identical to regular tuples over time.

#### A. Operators Classification

We distinguish four operator types according to recovery semantics: *arbitrary*, *deterministic*, *convergent-capable* and *repeatable*. Recovery plan type is determined by the most common operator in it.

An operator is *deterministic* if it produces the same output stream every time it starts from the same initial state and receives the same sequence of tuples on each input stream. There are three possible causes of non-determinism in operators: dependence on the time, dependence on the arrival order of tuples on different input streams, and use of non-determinism in processing (e.g., randomization).

A deterministic operator is (called) *convergent-capable* if it yields a convergent recovery when it restarts from an empty initial internal state and re-processes the same input streams, starting from an arbitrary earlier point in time.

A convergent-capable operator is *repeatable* if it capable of a repeating recovery when it restarts from an empty initial internal state and re-processes the same input streams, starting from an arbitrary earlier point in time and the operator produces identical tuples.

### III. HIGH AVAILABILITY PROTOCOLS

There are three basic approaches for achieving high availability in distributed systems: *process-pairs*, *logging* and *checkpointing*. In the following section, we introduce some of the current algorithms based on these approaches. Even when not all of them accomplish precise recovery, they can be extended to such level. Such extension includes for example duplicity

removal and protection from data loss. Data loss protection is mostly done by logging messages in output buffers until they are processed or stored by downstream (nodes further in the data flow) nodes. In case of failure, these stored messages are resent. Duplicity removal is protocol specific, so it is mentioned separately with each method.

#### A. Passive standby

Passive standby [6], [5] method is based on the *process-pairs* approach. There is a secondary node assigned to each primary node that receives state updates (checkpoints) from the primary node in regular intervals. In case of failure, the secondary node takes over computation from the last checkpoint. To achieve precise recovery, it is necessary to resend the data sent by upstream nodes, to recreate failed node state on the new node and ask the downstream nodes for delivered tuples, so they are not send for the second time.

The main advantage of this method is short recovery time consisting of reprocessing tuples received since last checkpoint and discovering tuples that were already send by the crashed node. However, the computing power of the regular computation is degraded, because (at least) half of the nodes are allocated as secondary nodes and communication is increased by sending regular checkpoints. Another possible slowdown is introduced when using synchronous backup (primary node does not send data until checkpoint is confirmed on secondary). When using asynchronous backup, data loss and inconsistent state can happen until logging of output buffers is introduced.

#### B. Active standby

Active standby is another version of the *process-pairs* approach [6], [5], [7]. Similarly to passive standby, each processing node has a dedicated secondary node. But unlike passive standby a secondary node does actively obtain and process same tuples as a primary node. Secondary node output is then logged out instead of send further downstream. Preventing duplicate messages by identifying the messages received by downstream nodes before sending the same messages computed by the secondary node is necessary to achieve precise recovery. Also, in case of non-deterministic operators, their decisions made on primary nodes have to be logged and sent to the secondary node, so it produces exactly the same results.

Minimal recovery time is the main advantage of the active standby over passive standby. That is because there is no need to reprocess data after failure, however at the cost of significantly higher communication, because all data must be sent also to secondary nodes. Another extra communication may introduce a queue trimming protocol, and sending of the decision logs in case of non-deterministic operators.

#### C. Upstream backup

Large run-time overhead is the main drawback of the process-pair approach (where at least half of the nodes are designated as backup nodes, thus not actively participating in computation). Upstream backup [6], [5] is designed for better use of distributed character of a stream computation. Upstream nodes (nodes against data flow) serve as backups for their downstream nodes by logging their output tuples. In case of

failure, a new node with empty state takes over computation and reprocesses stored tuples to get the same internal state as the failed node had.

Output buffers that backup tuples can grow in size equivalent to the size of tuples passing through that can be very space-demanding. To solve this inefficiency, the queue trimming protocol is introduced. It is based on the finding a minimal set of stored tuples that is necessary to restore a crashed node state. The delivery confirmation protocol is used to inform upstream nodes about tuples that were delivered. Delivery of each tuple is confirmed using 0-level confirmation to the sender of the tuple. After receiving 0-level confirmation, the node will know that the given tuple and all the proceeding tuples were delivered by recipient that send the confirmation. Once a tuple is confirmed by all recipients, the node determines last input tuple that was used to compute confirmed one, but it is not used to generate newer tuples anymore. If such tuple is found, it is confirmed to its sender using 1-level confirmation. Output buffers can be trimmed at the position of a tuple that received 1-level confirmation from all its recipients. Higher confirmation levels (by iteratively repeating confirmations) can be used to achieve better protection against multiple faults, however at the cost of less trimming efficiency, thus higher data space requirements.

Low extra bandwidth that is necessary for small confirmations and data transfers only during recovery is the main advantage of this method. However, this is at the cost of re-computation of many tuples during recovery. Also fail-free computation is slowed down when computing higher level confirmations that can be non-trivial.

#### D. Cooperative passive standby

Cooperative passive standby [8] is based on the *checkpoint* approach and on advantages of distributed computing (i.e., expandability, improved performance, etc.). Each computation is composed of computational units that are connected by data streams. Computational units are assigned to available hardware nodes to execute them. Traditional checkpointing of a more complex, or data intensive units is time demanding. To achieve better performance, this method splits computational units on each node into smaller parts called HA units. HA units are captured independently and then backed up on different nodes. This splitting divides the backup load of one node among multiple nodes.

Backing up each HA separately introduces finer granularity of the backup task; therefore, it can better fit into the spare time during computation (for example when pending for data) and increase overall system performance. Backup of each HA unit is done in two steps - capture and paste. During the capture, the update of the HA unit state is recorded and sent to the assigned backup node. During the paste, the node takes a received state updates for HA units backed up on it and apply delta updates into its copy of HA unit image. After paste, the initial node is notified, so it can schedule the HA unit for another checkpoint.

When a failure occurs, the backup nodes take over computations of the crashed node. During the takeover, the paste operations of unprocessed update messages take place and data are redirected to the backup nodes. Also to reflect the changes that occurred between backup and crash, the tuples

not included in the backups are resend from output buffers, so they can be reprocessed. Computation of HA units now continues on backup nodes.

This method has fast recovery time and the expected increase of workload on backup nodes is sufficiently small to preserve efficiency. If the crashed node becomes available again, it is added as a new empty node. HA units or their backups can be assigned to empty nodes or nodes with low workload as part of load balancing. HA units can be created and distributed between nodes automatically, so it can reflect actual situation and load balance.

#### IV. D-BOBOX

The Bobox [9], [3] is a parallel framework, which was designed to support development of data-intensive parallel computations. The main idea behind Bobox is to create a system that connects a large number of relatively simple computational components into a nonlinear pipeline while preserving transparency of the distribution logic to the authors of computational components. The pipeline is then executed in parallel, but the interface used by the computational components is designed in such way, that their developers do not need to be concerned with the parallel execution issues such as scheduling, synchronization and race conditions.

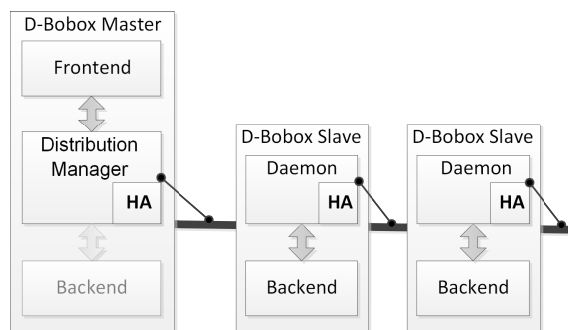


Figure 1. D-Bobox architecture. There is a single Master node that is responsible for task preparation and its distribution to slave nodes. The remote communication at the Master node is done via its Distribution Manager. Slave nodes contains Daemon parts that is responsible for the remote communication and tasks. The Distribution Manager and the Deamon are extended to provide HA. The Backend part on the Master node is optional.

D-Bobox [10] is an extension of the Bobox framework that adds support for distributed environment. This allows the framework to be used for tasks where local parallelism is not enough to achieve effectively fast computation. To preserve versatility of the framework, it is designed to run not only on specialized computational clusters, but on a common hardware too.

Base schema of the D-Bobox is described in Figure 1. The *master node* is responsible for creating an execution plan of the task and communication with the user that enters the task and monitors its computation. The master node also decides which other nodes will be participating in computation as *work (slave) nodes* and (typically) collects results. D-Bobox uses Bobox computation logic at each slave node and wraps it with remote communication and other necessary functionality needed for distributed environment. A remote communication

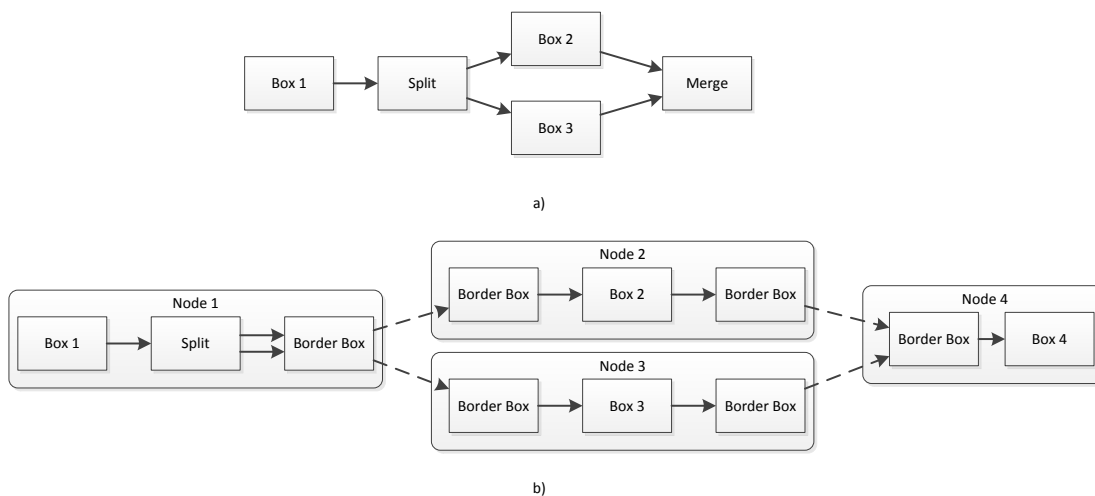


Figure 2. Sample of an execution extended into distributed environment using boundary boxes. a) Bobox plan for single node execution. b) D-Bobox plan utilizing four nodes. Original plan is split and extended by adding boundary boxes that manage remote communication (dashed).

is the most important extension during execution. It is implemented in the special boxes - border boxes that are added into the execution plan before slave nodes are initialized. Adding and configuring of border boxes is done primarily by the distribution control logic at the master node, according to the actual configuration and availability of nodes in cluster. Border boxes on a slave node are configured on request of the master node when necessary. Example extended execution plan is depicted at Figure 2.

## V. HA AND D-BOBOX

### A. Basic algorithms and D-Bobox

Each of the algorithms mentioned in Section III is applicable to D-Bobox system with different impacts to transparency, computation boxes requirements and changes to the framework itself. In the following discussion, we are focusing only on deterministic operators. Implementation of non-deterministic boxes breaks transparency requirement for each method, as they are required to log non-deterministic decisions and provide backup logic to achieve precise recovery.

Upstream backup represents a least blocking approach with minimum extra communication during fault-free computation. However, space requirements for storing output queues, recovery time and computation needed after the failure are typically quite high. Implementation of queue trimming protocols reduces these disadvantages at cost of higher communication and some computation slowdown during building and searching mappings between input and output tuples. When users are creating a new box, they have to implement these sometimes non-trivial mappings. That negatively impacts the transparency requirement and reliability of the framework. Reliability of the framework that depends on correct user implementation of the mappings is not suitable for our framework.

The active standby approach represents maximum transparency. It can be reflected by the computation plan used in D-Bobox by duplicating appropriate plan fragments and redirecting each of their outputs to the special communication box, located on the downstream node, so it is not affected

by the failure of the primary or the secondary node. This communication box receives tuples produced by the primary and secondary node; and forwards only tuples from primary node and stores tuples from the secondary node not yet produced by the primary node.

When a node failure occurs, the secondary node then becomes primary and continues in the computation. Then new secondary node is chosen and initialized by the new primary node current state. Computation speed at the primary node can differ from the secondary node, so after a failure, the communication box must correctly manage the data stream to preserve data consistency. When the primary node was ahead of the secondary, then the communication box must drop duplicate tuples produced by the secondary to prevent duplicate data. At the opposite situation, when the primary node was slower than the secondary node, then the communication box must send stored tuples to prevent gap in the data stream.

Effective computation power of active standby is halved since half of the nodes are reserved as backup nodes. Therefore, this approach is appropriate for problems, where very fast recovery time is more critical aspect than overall computation time. Since D-Bobox is oriented to be computation efficient, this approach is not appropriate to provide base HA functionality. However, it can be easily introduced for specific tasks that require very fast recovery instead of fast processing.

Passive standby approach also suffers from the same cutting of the effective computation performance as the active standby and is slower in recovery and during regular computation. Therefore, it is less practical than active standby. Another transparent approach is Cooperative passive standby that combines checkpointing and splitting backup to multiple smaller tasks and then distribute it between distinct nodes. Distribution of the smaller tasks increases backup performance, reduces recovery time and reduces work increase on backup nodes. Thanks to its distribution character and potential efficiency, we decided to use it as the base method for recovery from node failure to achieve HA in D-Bobox system. In the following subsection, we describe its integration in more detail.



## B. Integration of the High availability into D-Bobox

High Availability support that will handle node failures in D-Bobox is based on the cooperative passive standby method and it is located in new execution units called HA managers. HA managers are divided according to their specialization and location in the system to:

- *global HA manager* located at the primary node,
- *local HA managers* located at secondary (worker) nodes.

Each manager type handles different tasks: local HA managers perform local tasks that include local computation and backups stored on local node. The global HA manager is a global coordinator that assigns backup nodes for local HA units and handles recovery after failure detection.

1) *HA manager at the primary node*: is also called global HA manager. Its primary focus is to handle global tasks such as failure handling, assigning backup nodes to HA units and cooperation with the load balancing unit. An example of the distribution of backups of HA units to other nodes, as assigned by the global HA manager, is depicted in Figure 3.

Handling a node failure at the primary node is described in the Algorithm 1. Global HA unit notifies backup nodes to take over crashed node computation, reroute data and choose new backup nodes for restored HA units. After notifying upstream HA units, they resend data from output buffers to recreate the lost state not reflected in the last checkpoint. The increase of workload at a backup node after takeover is expected to be in acceptable boundaries because of the distribution of the work among multiple physically independent nodes.

A crashed node joins the set of nodes after recovery as an empty node and it can be dynamically assigned to backups or tasks during load balancing or after crash of another node. Global HA manager should support load balancing to achieve better performance of the computation and backups. When the system is highly unbalanced, then heavily loaded nodes cannot backup efficiently. They do not have spare time to backup, so they increase backup intervals that make backup more difficult and more costly, or they block computation often. On the other hand, idle nodes produce backups that can further slowdown loaded nodes. Dynamic load balancing increases the chance of evenly scheduling backups into idle CPU cycles when the computation is waiting (i.e., to receive new tuples). Moreover, processing backups more frequently reduces the amount of containing tuples in the backup, and less tuples have to be stored in the output buffers and recomputed during recovery.

2) *HA manager at the secondary node*: represents local manager that manage HA tasks of the current node as for example:

- monitoring neighbors availability (both upstream and downstream),
- administration of local HA units (for example splitting, merging),
- planning and performing of HA units backup (represented by the operation capture),

---

### Algorithm 1 Processing a node crash on the primary node

---

```

for all HAunit in crashedNode do
  backupNode ← getBackupNode(HAunit)
  backupNode.takeOver(HAunit)
  for all edge in HAunit.io do
    if isRemote and isInput then
      set_edge_target_to_backup
      resend_cached_tuples
    else if isRemote and isOutput then
      set_edge_source_to_backup
    else if isLocal and isInput then
      add_new_remote_edge
    end if
  end for
end for
for all backup in CrashedNodeBackups do
  find_and_set_new_backup_node
end for

```

---

- planning and performing of merge of the received updates from remote HA units into their local backups (represented by the operation paste).

Availability of neighbors is monitored by each node. Requests sent in regular intervals to test the availability are used when there is no communication among nodes at the time. If the node stops to respond, then after a defined amount of time (according to a preset time limit) is declared as crashed. The global HA manager is notified that node failure occurs and left to take care of the situation.

*Capture* and *paste* operations are two main operations providing backup functionality. During capture operation, the difference in HA unit state is captured and sent to the backup node. Paste operation on the backup node processes the received update messages by applying them to the local copy of HA unit state. Capture and paste operations of each HA unit are performed independently of each other, according to the used scheduling algorithm. Scheduling is performed locally on each node by the local scheduler that can implement different strategies to balance backup performance and computation performance.

Local HA managers support splitting and joining local HA units to balance granularity. Splitting is possible only if a HA unit consists of more than one computation box. Typically splitting a more complex HA unit results in dividing the backup overhead into a few smaller units that are easier to backup. Smaller HA units also pose smaller increase of the workload on the backup node, when it takes over the computation after recovery. On the other hand, a join operation is used to group multiple simple operations where an increase in the backup cost is smaller than the reduction of backup communication. For example, few HA units, each consisting of a simple box connected together into a pipeline, are good candidates to merge. In this case, we can suppose that the increase of backup complexity of the joined HA unit will be smaller (simple boxes, locality of the data) than the backup overhead of multiple HA units alone.

These changes of HA units size granularity have to be coordinated with the global HA manager. It must assign new

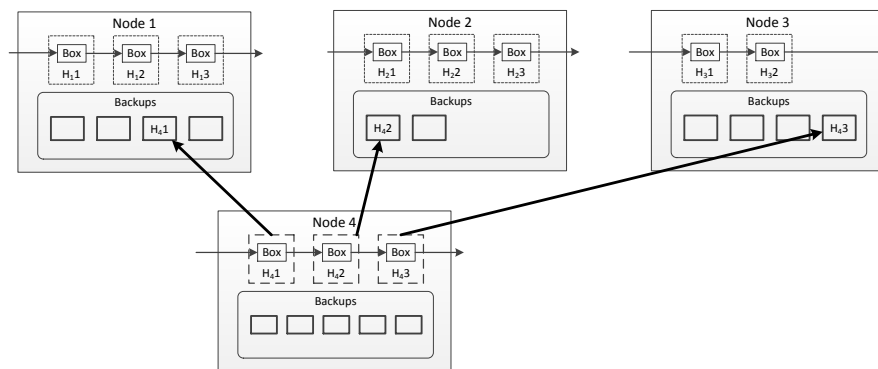


Figure 3. Example of the distribution of HA units demonstrated on the Node 4. Execution plan of the Node 4 is split into three HA units  $H_{4.1}, \dots, H_{4.3}$  that are backed up to the nodes 1-3 (backups are represented as boxes  $H_{4.1}, \dots, H_{4.3}$  in backup part of nodes 1 to 3).

backup nodes to each new HA unit that was created by the split operation or revoke backup role from nodes in case of joining of HA units.

A secondary node also contains a specific local communication manager that is part of special boundary boxes. Boundary boxes are special system boxes that represent endpoints for the remote communication to hide it from users developing regular computation boxes. Communication HA manager extends them by adding message logging and duplicity elimination. Outgoing messages are logged until the confirmation of their backup from HA units arrive. This is necessary to restore the computation state by computing these messages again after recovery from the last backup to reflect lost changes. Input border boxes have to be able to eliminate duplicity tuples that may be produced during recovery (crashed node may or may not produce some tuples that are not reflected in backup).

Extending D-Bobox with proposed managers provides high availability support to the framework. By using Cooperative passive standby as a core method that splits the node tasks into separate HA units distributed to different nodes, we get efficient recovery after a possible node failure. Another fine tuning of the HA units granularity according to actual computation states further improves the overall system performance. Smaller HA unit backups also pose acceptable increase of workload of the backup nodes after the recovery. The proposed approach also preserves transparency; the users creating applications should not be concerned with recovery methods.

## VI. CONCLUSION AND FUTURE WORK

In this paper, we introduced our distributed framework D-Bobox that is targeted on processing BigData (i.e., semantic databases). We presented a node failure problem in distributed data stream processing systems. Then we defined recovery types and summarized the main contemporary approaches to achieve high availability in such systems. We analyzed these approaches for their applicability in D-Bobox; we proposed an implementation of HA support in the framework. Using such HA support, the framework became capable of creating failure-resistant applications for data intensive computations in the distributed environment on a commodity hardware. The framework also preserves high level of transparency;

the users do not have to solve technical details concerning parallelism, distributed processing or high availability logic. In our future work, recovery support can be further extended by adding support of nondeterministic operators or adding new scheduling or by load balancing strategies.

## ACKNOWLEDGMENT

The authors would like to thank the GAUK project no. 472313 and SVV-2014-260100 and GACR project no. P103/13/08195S, which supported this paper.

## REFERENCES

- [1] M. Balazinska *et al.*, "The design of the borealis stream processing engine," in *Second Biennial Conference on Innovative Data Systems Research (CIDR 2005)*. CIDR, 2005, pp. 277–289.
- [2] D. Abadi *et al.*, "Aurora: a data stream management system," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM, 2003, pp. 666–666.
- [3] D. Bednarek, J. Dokulil, J. Yaghob, and F. Zavoral, "Bobox: Parallelization Framework for Data Processing," in *Advances in Information Technology and Applied Computing*, 2012, pp. 189–194.
- [4] Z. Falt, J. Dokulil, M. Cermak, and F. Zavoral, "Parallel sparql query processing using bobox," *International Journal On Advances in Intelligent Systems*, vol. 5, no. 3, pp. 302–314, 2012.
- [5] J.-H. Hwang *et al.*, "High-availability algorithms for distributed stream processing," in *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, 2005, pp. 779–790.
- [6] J.-H. Hwang, M. Balazinska *et al.*, "A comparison of stream-oriented high-availability algorithms," Brown CS, Tech. Rep., 2003.
- [7] M. A. Shah, J. M. Hellerstein, and E. Brewer, "Highly available, fault-tolerant, parallel dataflows," in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '04. New York, NY, USA: ACM, 2004, pp. 827–838. [Online]. Available: <http://doi.acm.org/10.1145/1007568.1007662>
- [8] J.-H. Hwang, Y. Xing, U. Cetintemel, and S. Zdonik, "A cooperative, self-configuring high-availability solution for stream processing," in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*. IEEE, 2007, pp. 176–185.
- [9] D. Bednarek, J. Dokulil, J. Yaghob, and F. Zavoral, "Data-flow awareness in parallel data processing," in *Intelligent Distributed Computing VI*, ser. Studies in Computational Intelligence, G. Fortino, C. Badica, M. Malgeri, and R. Unland, Eds. Springer Berlin Heidelberg, 2013, vol. 446, pp. 149–154.
- [10] M. Cermak, Z. Falt, and F. Zavoral, "D-bobox: O distribuovatelnosti boboxu," in *Informacne Technologie - Aplikacie a Teoria*. PONT s. r. o., 2012, pp. 41–46.

# Trustworthy Laboratory Automation

Jan Potthoff, Dominic Lütjohann, Nicole Jung

Karlsruhe Institute of Technology (KIT)

Karlsruhe, Germany

[name.surname]@kit.edu

**Abstract**—To ensure the quality of scientific data, the integrity and authenticity of the data has to be guaranteed. Due to the significance of the primary data, the integrity and authenticity of research data have to be ensured with their generation. Special measurement devices offer this possibility by an automated digital signing process. Unfortunately, these devices are rare. Therefore, a generic software application which is shown in this paper has been implemented. Furthermore, the solution has been adjusted to an existing laboratory automation system which is depicted as well. The combination of a new, web-based Laboratory Information and Management System (LIMS) with a new protocol for an authentication of electronic data shows that the establishment of an automated collection of secure data can be realized even without disturbing the researcher.

**Keywords**—*integrity and authenticity; sustainability; trustworthy data generation; data management*

## I. INTRODUCTION

In the field of natural sciences, data collections and tools for their valuation are of enormous importance as primary research data influence directly all experiment-driven interpretation [1]. Consequently, these primary research data form the basis of common knowledge. Nonetheless, the collection of scientific data sets –especially in academic labs– is very often not an automated process [4]. Up to now, the single researcher is responsible for the selection of the most important data, for the storage of these data and the researcher has to approve the integrity and authenticity of the information [2]. This traditional procedure of data acquisition will probably never be replaced by software-aided processes if the decision of the researcher is necessary.

In contrast, other processes as the acquisition of data generated by half-automated lab devices seem to be suitable for an automated collection, certification and management procedure. Examples of the latter operations are the analytical instruments that run with their own proprietary firmware giving standardized datasets that are the basis of qualitative and quantitative measurements [3]. Data collection of data produced by such devices means on the one hand the implementation of an automated procedure for the permanent actualization of the given research data (for each project, researcher, or group) and on the other hand the implementation of a reliable protocol for a doubtless statement on the integrity and authenticity of the gained datasets. Both challenges have been addressed and are

described herein via the presentation of selected examples (Section I to IV).

In Section II, the requirements of analytical devices for long-term preservation and the motivation for developing tools for trustworthy data management are depicted. A detailed example of the data generation process is added via description of an automated data collection process in Section III. Finally, the implementation of a ready to use software tool (Section IV) is exemplarily shown to demonstrate the applicability to gain trustworthy primary data in a lab environment.

## II. SCIENTIFIC DATA LIFECYCLE

In general, the research process is highly individual and differs from one scientific branch to another and from one researcher to another. However, the experimental scientific process flow may be roughly divided into five phases (planning, implementation, analysis, publication and archiving) [5]. In each of these phases, digital data is produced especially by measurement devices and the usage of computers.

### A. Measurement Devices and Scientific Data

The equipment of an exemplarily chosen chemical lab consists of three different groups of devices. They are divided into synthetic devices (as shakers, stirrers, heaters, microwaves and reactors), purification stations, e.g., preparative high-performance liquid chromatography (HPLC), medium pressure liquid chromatography (MPLC) and analytical devices as liquid chromatography mass spectrometry (LCMS), gas chromatography mass spectrometry (GCMS), nuclear magnetic resonance (NMR), ultra violet (UV), infra-red (IR), and Raman spectrometers. Whereas the devices in the first group are often standalone devices with almost no functionality for electronic data storage, the devices of group 2 and 3, namely the purification stations and the analytical devices, contain intern intelligence for the acquisition and management of electronic information. Therefore, the investigations on the storage and authentication of scientific data focus initially on the analytical devices which process all available information and which are of highest interest as they deliver the final data of the research projects.

If possible, the data format is chosen by its creator. Because of several data sources, e.g., software programs or measurement devices, as described above, a free choice is not always possible, proprietary data formats have to be

accepted and the data migration becomes difficult. Based on several catalogues of criteria, the data formats can be evaluated according to their suitability for long-term preservation. It is important to keep in mind that the choice of data format should respect the actual situation but also the archival process [6].

### B. Trustworthy Data Management

Data processing systems have to meet several requirements to ensure the quality and protection of data. For example, the IT-Grundschutz Catalogue of the Federal Office for Information Security (BSI) defines four core requirements namely availability, confidentiality, integrity and authenticity referring to information security [7]. These criteria are very important for all processes that are based on a high level of trustworthiness such as archival systems. In this case, the availability, confidentiality, integrity and authenticity have to be ensured for a long time. Therefore, the reference model Open Archival Information System (OAIS) defines essential functions and processes to guarantee these requirements [8].

As most of the available data will not only be generated and archived but will also be used, the data have to be managed by specific programs or individual folder structures. The type of data management depends on the application, the requirements of the organization and the individual requirements of the user. For example, in research, Electronic Laboratory Notebooks (ELN) or Laboratory and Information Management Systems (LIMS) are used for the data management [9]. These paperless lab management alternatives offer many advantages in contrast to the documentation on paper, but they suffer from one important disadvantage: changes in digitally stored data are not detectable.

Within the DFG project “BeLab” (probative electronic laboratory notebook) an interdisciplinary work group analyzed how accessibility, completeness, integrity, authenticity, readability and interpretability of ELNs can be ensured in the long term [26]. The result of the project is a service (BeLab system) which ensures the integrity and authenticity of the submitted data. As different research areas use individual scientific tools in their processes, the BeLab system has been designed as a generic data verification system which uses multiple ingress, verification and egress modules. By using the service, the provability of scientific data in digital archives is profoundly enhanced [10]. In [11], a generic solution for data management according to the Good Scientific Practice (GSP) is depicted as a further result of the BeLab project. By this application the scientists are able to archive their data with respect to the GSP. The data can be managed by a Graphical User Interface (GUI) which offers the possibility to collect files to be archived or to check out archived files and edit them. Additionally, metadata which refer to the structure of data, e.g., project, ELN and general container id can be added. These metadata can be indicated for each added file in the data section as well. Examples for such additional descriptive information are document title, author and creation date. This generic solution prepares the managed data for a trustworthy

archiving. It shows that the integrity can be easily implemented within the data management environment. Regarding the long-term preservation, the data can be submitted to the BeLab system.

### C. Related Work

The integrity and authenticity of scientific data must be ensured in regards to several regulations [12][13]. In addition, all research fields have to guarantee a sustainable archiving of this data [14]. To foster and to assess the trustworthiness of archival systems, further research projects have addressed the issues of availability, confidentiality, integrity and authenticity, as described in the previous section [15][16].

To ensure the integrity and authenticity, the research process flow has to be considered as well. Current available systems are tailored for specific device manufacturers usually commercially available as integrated and monolithic lab automation solutions [17]. Customization and expansion, e.g., the integration of new devices or devices from different vendors, requires drivers and software components specifically developed for the platform and are usually not usable in other environments.

Open Source solutions [18] for lab automation allow flexible implementation of new devices, but are usually not prepared for data acquisition in a dynamic lab infrastructure which requires frequent changes in configuration such as laboratories in academic facilities. Furthermore, installation and maintenance requires investments in server infrastructure, as long as data ownership needs to be considered and commercial cloud services cannot be taken into consideration.

## III. LABORATORY AUTOMATION

The automated data generation process is the key requirement in data-driven laboratories. Existing devices and computer systems need to be integrated into unified views of the overall data structure of a laboratory to allow scheduling of tasks, remote operation of devices, sample tracking, data management, and archiving.

### A. Practical Requirements

Many laboratory devices are equipped with a command and control personal computer system, which is connected via ethernet connection or other serial interfaces to the device. To operate the device, vendor-specific software needs to communicate with the device using device drivers and command sets which often follow proprietary protocols. To integrate these kiosk-like systems in an automated laboratory, an overlay software is required, which does not need deep modifications on operating system and application level. Furthermore, it should not be necessary to adjust the data storage routines and locations, to access the created datasets in a LIMS.

### B. Webbased Device Control

Once samples are loaded onto a device, the data acquisition process takes time, depending on the selected method. To allow users to operate devices remotely, e.g.,

from different workplaces and different mobile devices, and observe the output values and analysis status, the user interface needs to be shared on a web-based platform. The experimenter who operates the device also needs control over the generated datasets from a centralized platform to integrate the contained results into a research and results database. Beginning from this step, identification of the user is possible and therefore the ownership of datasets can be determined. This allows data policies to be applied at the time of data collection and enables ubiquitous access.

#### IV. TRUSTWORTHY DATA COLLECTION

The amount of digital data is constantly increasing. To be able to manage these data, special software and hardware solutions have been developed over the last years. Procedures to ensure the integrity and authenticity of the data in paper documentations have already been established but these regulations have to be transferred to the electronic datasets as well. To ensure the trust in new, automated technologies, specific requirements have been described. The obligation to preserve records also applies for these digital data. This leads to several requirements regarding the digital long-term preservation as well. One of these requirements is the integrity of all produced data: "Integrity considers all possible causes of modification, including software and hardware failure, environmental events, and human intervention." [19]. The combination of these tools will end in a trustworthy archive, which should prevent the quality of data. In addition to that, the quality of data depends on its generation, processing and preparation. To ensure the integrity and quality of all given information, the overall process of their generation has to be analyzed.

##### A. Practical and Legal Requirements

The importance of the verification and integrity of data is always desirable and in many cases absolutely mandatory. Due to fast proceedings of computing and digital processes that find their way into almost all areas of life, the legislative organs worldwide have to be concerned with regulations upon these new paperless developments. Whereas, in general, all data can be easily manipulated, digital data are even more prone to undesired alteration as changes –even in much larger scale– can be easily made without being visible to others. In the year 2001, the German law responded to these developments and special bills have been developed for digital data. By these bills, the evidence of digital documentation is regulated based on the digital signature. This signature can be used to prove the integrity of digital data and the authenticity can be shown by the corresponding certificate [20]. A further opportunity to validate the integrity of digital data is the qualified digital time stamp, which is required in several organizations and which can be used to prove the existence of a document. Whereas other alternatives offer very similar opportunities, the processing of a qualified digital stamp is the only procedure that has been accepted by the German law [21].

To ensure a high probative force of digital data, the integrity and authenticity has to be ensured as early as possible, which usually means at the same time as the data is

generated [22]. A lot of analytical devices which generate digital data are used for experimental research, so that the best possibility would be to ensure the integrity and authenticity by these devices. This kind of hardware-triggered solution could be implemented as integrated software package within the analytical device. A few of these analytical devices with implemented digital signing procedure are available today, but adopting this solution will result in the dependency on the manufacturer. A generic approach should be addressed giving a solution that should be independent of the used devices, the research area and the used hardware. The only restriction to be made is that it will be assumed that the analytical device is connected to a computer. Due to the independency of the research area, the solution has to be adaptable to further requirements of individual use cases.

Due to the juristic requirements, as previously described, the solution should use digital signatures to ensure the integrity of data. The development of such a signature trail is always the result of a compromise: As the researcher should not be affected by the signing process, the digital signature should be added, on the one hand, in an automatically manner. On the other hand, the automatic signing process decreases the probative value of the signature. Therefore, the process has to be associated with the person [23].

Because a generic approach will be separately implemented from the analytical devices, the time between data generation and integrity protection is of prime importance. The period of time should be as short as possible. In addition, other organizational measures, e.g., room entry controls, can be also implemented to ensure the data integrity. The implemented safety measures should be transparent as well as the functioning of the solution to reach the trust in the solution.

To keep the reached probative value, a trustworthy archive should be used at the end. Because the measurement data should not be altered, the archival process can be directly started after the data generation. By doing this, the data sharing and usage is reached as well.

##### B. Protected Data Collector

A secure data generation can be reached by digitally signing analytical devices. These devices have an internal cryptographically chip that calculates the signature. By doing this, the data is signed before leaving the device and a data manipulation can only be reached by a manipulation of the device. By sealing the devices, the manipulation can be proven and the integrity and authenticity can be validated by the signature and the corresponding certificate.

According to the practical and legal requirements, as described in Section IV-A, and to become more generic, a software solution (hereafter called data collector) has been developed to offer the functions of data integrity and authenticity for individual analytical devices. In a first step, a generic interface between device and the data collector, e.g., a folder for the data transfer, has to be defined. Using such a folder, analytical devices store their data via a gateway that can be monitored by the data collector. If the data collector

detects new files, it will execute all necessary steps to ensure the integrity and authenticity of the processed data.

A particular challenge is the definition of the time of archiving. For this purpose, one folder can be monitored by several individual modules. For example, a module monitors the number of files in the corresponding folder. If a defined number of files are detected, the archival process will be started. These modules can be further specified in the system configuration offering the possibility for combinations of different modules and the integration of individual modules.

If a new file is detected, the corresponding hash value will be calculated and stored in the main memory. From that on, the integrity of the detected file is temporarily secured. After each process, the individual modules check the archiving condition and the results will be logically combined to one outcome. All hash values will be calculated again prior to the archival process and they will be compared with the values in the main memory. If no changes have been detected, the data will be digitally signed by the data collector. Before this, the detected files will be prepared for the long-term preservation which includes the package of the analytical data into a data container, called Universal Object Format (UOF) [24]. This container which includes integrated metadata has been developed by a German research project of long-term preservation [14]. The metadata format is based on the Metadata Encoding and Transmission Standard (METS) [25] that includes attributes of file, e.g., filename and hash value. With this information, the integrity can be proven at any time. Whereas this procedure still allows the manipulation of files and hash values, the metadata file is digitally signed so that all changes can be proven. By doing this, the legal requirements, depicted in Section IV-A, are addressed.

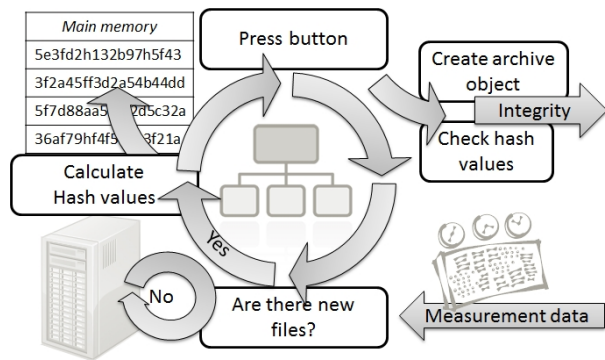


Figure 1. Working process of data collector

After finishing the process mentioned above, the data will be submitted to an archive interface. This interface checks the data according to the requirements for long-term preservation and its probative value. For example, existing digital signatures will be validated again. In addition, a specified module checks the completeness, e.g., filename dependencies are analyzed.

The data collector can be used in the data management process, as described in Section III, if it is adjusted according to the time of archiving. Due to an architecture where an

individual amount of files belongs to one output directory and several subfolders belong to one measuring, a corresponding archiving module cannot be implemented for these data management purposes. As a solution, one root folder is monitored in which a new subfolder will signal new experiments that have to be monitored. After this initial procedure, all files and subfolders, which are stored in this folder, will be noticed by the data collector and the integrity is secured as previously described.

The archiving starting point can be signaled by the user with a small GUI. This GUI contains a button which is inactive if no new subfolders are noticed. If a new folder is noticed, the button will become active and the user can start the archival process at any time. The process of securing the integrity and archiving by the data collector is depicted in Figure 1.

By pressing the button, all noticed subfolders of the root folder will be prepared for an archive object and will be submitted to an archive interface as previously described. After that, the button will be disabled again until new subfolders are founded.

Naturally, the researcher can modify the primary analytical data after the data has been submitted to the archival system. For this purpose the archival process has to be adapted. The data collector will notice the changes of the corresponding files. If files, which have been already archived, are changed and the subfolder is noticed again, the archived data must be updated. For this the archival process can be restarted again by the user. To handle all versions of the data, all IDs received from the archival system and the corresponding folder names are stored. If the folder name already exists in this list, the update function will be used in combination with the corresponding ID. This way, the integrity of measurement data is guaranteed and all research results are traceable.

With this solution, the measurement device must not calculate the required digital signature by itself. Thereby the data integrity assurance can also be used for devices with a fast data processing. Additionally, the archival process is separated from the data processing so that the scientist is not affected.

### C. Safety Aspects

The highest probative value is reached by digital measurement devices which offer the possibility of internal digital signatures. On the other hand, this function is not implemented in many devices yet. Therefore, a mechanism, as described in previous Section, is needed which can be used in combination with each measurement device. However, some security gaps have to be kept in mind.

The first challenge is the data transfer from the analytical device to the connected computer. During this procedure, manipulations are most likely in particular if the device is connected via network. In this case, a secure data transfer protocol like https has to be used. If the files are stored, the data collector will need a short time until it detects the new files and a security gap results. In consequence, data manipulation may occur between the finishing of the file and its detection. This fact can be addressed technically by

minimizing the time gap. If operating system-based solutions are used, the time gap will reduce to a few milliseconds. By doing this, a data manipulation is improbable.

After the detection of new data the hash values will be calculated by the data collector and stored in the main memory. Theoretically the values in the memory can be manipulated on a higher level. This manipulation would mean a high criminal energy. Generally, the solution is not developed to prevent any faults but rather to give a solution to demonstrate the appropriate work of the researcher. In the final step, a manipulation can be made during the data transfer to the archival system but the used archive interface is based on the https protocol so that a secure data transfer is given.

On the juridical side, the focus is on how probable a manipulation is. If a network is used, the mistrust will be high even if the data transfer is encrypted, whereas a separated room with controlled human access leads to a higher trust. Data processing is not the critical point if a secure environment can be established. Generally, digital data can be manipulated easily. Therefore, particular laws have been established by the German legislation to rule the usage of digital signature and to get a probative force. Therefore, the submitted data will be automatically signed by the data collector. According to the usage of an automatically process for digital signatures, it has to be noted that the procedure has to be connected to a person, which means that a person has to start and initialize the system with its corresponding certificate [23]. Only then a digital signature is comparable with a handwritten signature according to the German law. The same applies to signing measurement devices.

## V. CONCLUSION AND FUTURE WORK

The implemented software application data collector aims at providing a generic solution for the guarantee of the data integrity and authenticity. In comparison with digital signing measurement devices some safety risks have to be accepted. However, these risks can be reduced by administrative measures. The software application has been adjusted for the depicted workflow. It can be seen that the adjustment of the archiving condition was difficult because of individual data structures of the lab devices.

The depicted software solution fulfills the requirements of the German electronic signature law and the regulations of the GSP. For general usage, further regulations have to be considered.

The workflow in chemical labs is only one example for the need of professional data handling but forms an ideal model for the implementation of an automated data collector. Most of the processes in chemical labs underlie standardized procedures and the main datasets consist of values that have been acquired from well-defined, standardized instruments. In order to identify these processes in a chemical lab, the dataflow in an organic working group has been investigated. Advanced analytical instruments as NMR-spectrometers, chromatography-mass spectrometers (liquid and gas chromatography, HPLC/GC) and electron ionization (EI)-mass spectrometers were determined as devices with high

priority for the definition of experimental conclusions and for the confirmation of the results in publications. The implementation of a data collector covering results that have been processed by these instruments was starting point of a new interdisciplinary project.

## REFERENCES

- [1] J. G. Frey, "Dark Lab or Smart Lab: The Challenges for 21st Century Laboratory Software," *Org. Process Res. Dev.* 8, 2004, pp. 1024-1035.
- [2] C. L. Bird and J. G. Frey, "Chemical information matters: an e-Research perspective on information and data sharing in the chemical sciences," *Chem. Soc. Rev.* 42, 6754.
- [3] R. Bramley, K. Chiu, T. Devadithya, N. Gupta, C. Hart, J. C. Huffman, K. Huffman, Y. Ma, and D. F. McMullen, "Instrument Monitoring, Data Sharing, and Archiving Using Common Instrument Middleware Architecture (CIMA)," *J. Chem. Inf. Model.*, 2006, 46 (3), pp 1017-1025.
- [4] N. Jung and D. Lütjohann, "Chemie auf der Spitze des Eisbergs: Zu viele Forschungsdaten gehen bislang unter!," *Chemie in unserer Zeit*, vol. 47, 2013, pp. 334-335, DOI:10.1002/ciuz.201390062
- [5] S. Hackel, P.C. Johannes, M. Madiesh, J. Potthoff, and S. Rieger, "Scientific Data Lifecycle – Beweiswerterhaltung und Technologien," *Proc. 12. Deutscher IT-Sicherheitskongress (BSI-IT-SEC 2011)*, SecuMedia, 2011, pp. 403-418.
- [6] A. Brown, "Digital preservation guidance note: Selecting file formats for long-term preservation," 2006.
- [7] Federal Office for Information Security (BSI), "IT-Grundschutz Catalogue," 2005, <https://www.bsi.bund.de/EN/Topics/ITGrundschutz/itgrundschutz.html> 11.02.2014.
- [8] CCSDS, "Reference Model for an Open Archival Information System (OAIS)," CCSDS, 2002.
- [9] M. Rubacha, A. K. Rattan, and S. C. Hosselet, "A Review of Electronic Laboratory Notebooks available in the market today," *JALA*, vol. 16, Feb.2011, pp. 90-98, doi:10.1016/j.jala.2009.01.002.
- [10] J. Potthoff, S. Rieger, and P. C. Johannes, "Enhancing the Provability in Digital Archives by Using a Verifiable Metadata Analysis Web Service," *Proc. 7<sup>th</sup> ICIW 2012*, 2012, pp. 112-117.
- [11] J. Potthoff, M. Walk, and S. Rieger, "Data Management According to the Good Scientific Practise," *Proc. 5<sup>th</sup> DBKDA 2013*, 2013, pp. 27-32.
- [12] Deutsche Forschungsgemeinschaft, "Recommendations of the Commission on Professional Self Regulation in Science - Proposals for Safeguarding Good Scientific Practice," January, 1998, [http://www.dfg.de/en/research\\_funding/legal\\_conditions/good\\_scientific\\_practice/](http://www.dfg.de/en/research_funding/legal_conditions/good_scientific_practice/) 26.02.2014.
- [13] OECD, "OECD Principles on Good Laboratory Practice," Paris, 1998, <http://search.oecd.org/officialdocuments/displaydocumentpdf/?doclanguage=en&cote=env/mc/chem%2898%2917> 26.02.2014.
- [14] R. Altenhöner, "Data for the future: The German project 'Co-operative development of a long-term digital information archive' (kopal)," *Library Hi Tech*, Vol. 24 Iss: 4, 2006, pp. 574-582, doi:10.1108/07378830610715437.
- [15] Trustworthy Repositories Audit & Certification (TRAC), "Criteria and Checklist, Center for Research Libraries," OCLC Online Computer Library Center, 2007.
- [16] N. Beagrie et al., "Trusted Digital Repositories: Attributes and Responsibilities," *RLG-OCLC Report*, 2002.
- [17] Agilent OpenLAB, <http://www.chem.agilent.com/en-US/products-services/Software-Informatics/OpenLAB-CDS-Chemstation-Edition/Pages/default.aspx> 26.02.2014.

- [18] Bika LIMS, <http://www.bikalabs.com> 26.02.2014.
- [19] T. Malone, G. Blokdijk, and, M. Wedemeyer, "Itil V3 Foundation Complete Certification Kit-Study Guide Book and Online Course," Lulu. com, 2008.
- [20] S. Mason (Ed.), "International Electronic Evidence," London, UK: BIICL, 2008.
- [21] D. Huhnlein, "How to qualify electronic signatures and time stamps," Public Key Infrastructure, Proceedings, Lecture Notes in Computer Science, 2004, pp. 314-321.
- [22] J. Potthoff, S. Rieger, P.C. Johannes, and M. Madiesh, "Elektronisch signierende Endgeräte im Forschungsprozess," Proc. D-A-CH Security 2011, syssec, 2011, pp. 44-55.
- [23] A. Roßnagel, S. Fischer-Dieskau, "Automatisiert erzeugte elektronische Signaturen," MMR 2004; S. 133 – 139.
- [24] T. Steinke, "The Universal Object Format – An Archiving and Exchange Format for Digital Objects," in Research and Advanced Technology for Digital Libraries, Springer Berlin, 2006, pp. 552–554.
- [25] Digital Library Federation, "<METS> Metadata Encoding and Transmission Standard: Primer and Reference Manual," Version 1.6 Revised, 2010, <http://www.loc.gov/standards/mets/> 26.02.2014.
- [26] BeLab project, <http://www.belab-forschung.de> 26.02.2014.



# Toward a New Approach of Distributed Databases Design and Implementation

Hassen Fadoua

LIPAH

FST, University of Tunis El Manar

Tunis, Tunisia

hassen.fadoua@gmail.com

Grissa Touzi Amel

LIPAH

FST, University of Tunis El Manar

Tunis, Tunisia

amel.touzi@enit.rnu.tn

**Abstract**—Nowadays, with the development of data and storage of large volumes of distributed and heterogeneous data, Distributed Database Management System (DDBMS) have become essential to most Information Systems (IS). Unfortunately, the designer of Distributed Databases (DDB) has been so far facing several problems, namely, 1) the DDB design is not a simple task and should take into account several constraints and choose accordingly best strategy of fragmentation, allocation and replication of data and 2) DDB implementation should allow the final user to work within a centralized DB, which is not provided directly by the existing DDBMS. To sort out this problem, we suggest in this paper a new approach to help in the DDB design and implementation, which focuses on setting up a layer in the existing DDBMS which will provide 1) Graphical interface to define different sites geographically distributed and 2) Creation of different types of fragmentation, allocation and duplication while validating each step of the process. The system will automatically generate SQL scripts of each site regarding its initial configuration. The so implemented approach reduces the designer's duty by taking in charge the complex distribution validation and heavy manual scripts writing.

*Keywords*-distributed databases; fragmentation; fragment allocation; replication.

## I. INTRODUCTION

The end of the last century was marked by a significant change in information technology. This evolution is mainly characterized by large volumes of data increasingly important, distributed and heterogeneous information, and more exacting users toward system vendors and solutions. Design and use of distributed database has risen significantly.

Unfortunately, the existing DDBMS have several constraints: 1) they do not have an integrated component which ensures the automatic distribution of the initial centralized database, and 2) Fragmentation, replication and allocation are manual operations delegated to administrators. The designer is required to ensure the compliance of the distribution with the validation rules.

Consequently, the implementation of a DDB has never been an easy task especially when dealing with huge models and while trying to meet the high user's expectations. While looking into the constraint's causes by these systems, the most important requirements are to preserve data integrity and their continuous availability, even though the central site

has been removed, in addition to their transparency for the final user.

In this context, we can refer to the works of Rim [11] and Hassen [7] who suggested an expert system to help the DDB design. These tools are rather restricted to suggesting data distribution on each site, regardless of the heavy task left to the designer to implement this DDB on different sites or the validation process of fragmentation if the user decides to change its design in response to new needs.

In this paper, we propose a new approach to assist DDB design and implementation. This approach is validated through designing and implementing an assistance tool which provides a graphical interface for different types of fragmentation, allocation and replication along with validation at each step of the process. Then, the system will automatically generate SQL [3] scripts of each site regarding its initial configuration. We have proved that the proposed tool can be implemented as a layer to any existing DDBMS.

This paper includes five sections. Section 2 presents an example of DDB design, illustrating the design problems. Section 3 presents our motivation for this work. Section 4 presents our new approach to assist the DDB design and implementation. Section 5 presents the validation of our approach by providing the platform called DDB-Helper. Section 6 provides an evaluation of this work against existing approaches. We finish this paper with a conclusion and a presentation of some future works.

## II. PROBLEM OF DDB DESIGN

We define a distributed database (DDB) as a collection of multiple, logically interrelated databases distributed over a computer network [2].

A distributed database management system (distributed DBMS) is then defined as the software system that permits the management of the distributed database and makes the distribution transparent to the users [9]. As examples of DDBMS, we can mention: Oracle [5], MySQL [12], Ingres [10], Cassandra [4] and F1 [8].

The design stage of a distributed database must take into consideration a number of constraints, usually quite difficult to balance. This approach should be based on the description of the real world, the needs of the user and his frequent queries. The purpose of this section is to show through an example the difficulties that can meet the user in the design of its DDB.

A. Distributed Database Design

To set the local conceptual schema for each site, the designer should follow the steps:

(i) Fragmentation of the different relations: a relation can be divided into a number of sub-Relations, called fragments, allocated to one or more sites. There are two types of fragmentation: horizontal and vertical. The horizontal fragments are subsets of tuples and vertical fragments are subsets of attributes of relations

(ii) Correctness Rules of Fragmentation: The designer must check the three Correctness rules of fragmentation; Completeness, Reconstruction and Disjointness.

(iii) Definition of the allocation of fragment: This definition is carried out strategically, to ensure the locality of references, an enhanced reliability and availability, acceptable performance, a balance of storage capacity and costs, and communication costs reduced. Four allocation strategies exist, depending on the available data: centralized (single centralized database), fragmented (fragments are assigned to a site), full replication (a full copy of the database is maintained at each site) and selective replication (a combination of the other three).

B. DDB Design Example

In this section, we present an example of a DDB design. Three institutions of the University of Tunis El Manar: National Engineering School of Tunis (ENIT), Faculty of Mathematical, Physical and Biology Sciences of Tunis (FST) and Faculty of Economics and Management of Tunis (FSEGT) have decided to pool their libraries and service loans, to enable all students to borrow books in all the libraries of the participating institutions. Joint management of libraries and borrowing is done by a database distributed over 3 sites (Site1 = ENIT, Site2 = FST and Site3 = FSEGT). The global schema is described in Table II.

Managing this application is based on the following assumptions:

- i. An employee is assigned to a single site.
- ii. A student is enrolled in a single institution, but can borrow from all libraries.
- iii. A book borrowed from a library is returned to the same library.
- iv. The nb\_borrow field of STUDENT relation is used to limit the number of books borrowed by a student simultaneously in all libraries. It is updated at each loan and each return, regardless of the lending library.
- v. Each institution manages its own students.
- vi. Each library manages its staff and works it holds.

TABLE I. CENTRALIZED DATABASE SCHEMA

EMPLOYEE (NSS, FName, LName, Address, Status, Assignment)
STUDENT (NCE, FName, LName, Address, Institution, Class, nb_borrow)
BOOK (Id_book, Title, Editor, Year, Area, Stock, Website)
AUTHOR (Id_book, Au_lname, Au_fname)
LOAN (Id_book, NCE, date_borrowing, return_date)

An uninitiated designer in the concept of DDB can ask the following questions:

- i. How to determine the relationships that must be broken and the ones which will be duplicated?
- ii. In case of fragmentation, how to choose the attribute of fragmentation?
- iii. How to choose the allocation of fragments of a relationship and according to which strategy?

In this section, we merely describe design steps of our initial database. The aim of our approach is to provide a tool to help in the design of a DDB.

1) First Step: Relations Fragmentation

Relation EMPLOYEE:  
 EMPLOYEE\_ENIT =  $\Pi_{NSS, \text{fname}, \text{lname}, \text{Address}, \text{Status}}(\sigma_{\text{Assignment} = \text{'ENIT'}}(\text{EMPLOYEE}))$   
 EMPLOYEE\_FST =  $\Pi_{NSS, \text{fname}, \text{lname}, \text{Address}, \text{Status}}(\sigma_{\text{Assignment} = \text{'FST'}}(\text{EMPLOYEE}))$   
 EMPLOYEE\_FSEGT =  $\Pi_{NSS, \text{fname}, \text{lname}, \text{Address}, \text{Status}}(\sigma_{\text{Assignment} = \text{'FSEGT'}}(\text{EMPLOYEE}))$   
 Relation STUDENT

1) Vertical Fragmentation is applied to the STUDENT table  
 STUDENT\_Biblio =  $\Pi_{NCE, \text{lname}, \text{fname}, \text{Nb\_borrow}}(\text{STUDENT})$   
 STUDENT\_Inst =  $\Pi_{NCE, \text{lname}, \text{fname}, \text{Address}, \text{Institution}, \text{Class}}(\text{STUDENT})$   
 2) Then we applied a horizontal fragmentation on the table STUDENT  
 STUDENT\_ENIT =  $\Pi_{NCE, \text{lname}, \text{fname}, \text{Address}, \text{Class}}(\sigma_{\text{Institution} = \text{'ENIT'}}(\text{STUDENT}))$   
 STUDENT\_FST =  $\Pi_{NCE, \text{lname}, \text{fname}, \text{Address}, \text{Class}}(\sigma_{\text{Institution} = \text{'FST'}}(\text{STUDENT}))$   
 STUDENT\_FSEGT =  $\Pi_{NCE, \text{lname}, \text{fname}, \text{Address}, \text{Class}}(\sigma_{\text{Institution} = \text{'FSEGT'}}(\text{STUDENT}))$   
 Relation BOOK  
 BOOK\_ENIT =  $\Pi_{\text{Id\_book}, \text{Title}, \text{Publisher}, \text{Year}, \text{Domain}, \text{Stock}}(\sigma_{\text{Site} = \text{'ENIT'}}(\text{BOOK}))$   
 BOOK\_FST =  $\Pi_{\text{Id\_book}, \text{Title}, \text{Publisher}, \text{Year}, \text{Domain}, \text{Stock}}(\sigma_{\text{Site} = \text{'FST'}}(\text{BOOK}))$   
 BOOK\_FSEGT =  $\Pi_{\text{Id\_book}, \text{Title}, \text{Publisher}, \text{Year}, \text{Domain}, \text{Stock}}(\sigma_{\text{Site} = \text{'FSEGT'}}(\text{BOOK}))$   
 Relation AUTHORS  
 AUTHOR\_ENIT = AUTHORS  $\bowtie$  BOOKENIT  
 AUTHOR\_FST = AUTHORS  $\bowtie$  BOOKFST  
 AUTHOR\_FSEGT = AUTHORS  $\bowtie$  BOOKFSEGT  
 Relation LOAN  
 LOAN\_ENIT = LOANS  $\bowtie$  BOOKENIT  
 LOAN\_FST = LOANS  $\bowtie$  BOOKFST  
 LOAN\_FSEGT = LOANS  $\bowtie$  BOOKFSEGT

2) Second Step: Checking the correctness of the fragmentation

For each fragmentation, we must check: The completeness aspect, reconstruction and disjoint. We present the following reconstruction aspect that seems to be the most important and most critical.

i. EMPLOYEE relation's reconstruction  
 Ti is a relationship with a single attribute, the attribute assignment. The value of this attribute is i. The reconstruction of the starting relation EMPLOYEE can be done by a union (U) of all the EMPLOYEE fragments on each site and the selection (x) of the assignment attribute of Ti (column Assignement).

$$\text{EMPLOYEE} = \cup_i(\text{EMPLOYEE}_i \times T_i)$$

ii. STUDENT's relation reconstruction is done in several steps:

Ri is a relation with a single attribute, the Institution. The value of this attribute is i.

$$STUDENT_{Inst} = U(STUDENT_i \times R_i)$$

After  $STUDENT_{Inst}$  reconstruction, the initial relation can be obtained by a join ( $\bowtie$ ) of  $STUDENT_{Inst}$  and  $STUDENT\_BIBLIO$  duplicate fragment.

$$STUDENT = STUDENT\_BIBLIO \bowtie STUDENT_{Inst}$$

iii. BOOK's relation reconstruction

$S_i$  is a relation having a single attribute, the attribute site.

The value of this attribute is i.

$$BOOK = U_i(BOOK_i \times S_i)$$

iv. AUTHORS' relation reconstruction

$$AUTHORS = U_i(AUTHORS_i)$$

v. LOAN's relation reconstruction

$$LOAN = U_i(LOAN_i)$$

3) Step Three: Defining an allocation scheme for each site

Considering the hypotheses provided by the user, we decided to duplicate the  $STUDENT\_BIB$  table with a synchronous update. The resulting local schema is as described in Table III, Table IV, and Table V.

TABLE II. SITE 1 LOCAL SCHEMA: ENIT

EMPLOYE_ENIT (NSS, FName, LName, Address, Status)
STUDENT_ENIT (NCE, FName, LName, Address, Class)
STUDENT_BIB (NCE, FName, LName, Nb_borrow)
BOOK_ENIT (Id_book, Title, Editor, Year, Field, Stock)
AUTHOR_ENIT (Id_book, FName_author, LName_author)
LOAN_ENIT (Id_book, NCE, borrow_gdate, return_date)

TABLE III. SITE 2 LOCAL SCHEMA: FST

EMPLOYE_FST (NSS, FName, LName, Address, Status)
STUDENT_FST (NCE, FName, LName, Address, Class)
STUDENT_BIB (NCE, FName, LName, Nb_borrow)
LOANS_FST (Id_book, Title, Editor, Year, Field, Stock)
AUTHOR_FST (Id_book, FName_author, LName_author)
LOAN_FST (Id_book, NCE, borrow_gdate, return_date)

TABLE IV. SITE 3 LOCAL SCHEMA: FSEGT

EMPLOYE_FSEGT (NSS, FName, LName, Address, Status)
STUDENT_FSEGT (NCE, FName, LName, Address, Class)
STUDENT_BIB (NCE, FName, LName, Nb_borrow)
BOOK_FSEGT (Id_book, Title, Editor, Year, Field, Stock)
AUTHOR_FSEGT (Id_book, FName_author, LName_author)
LOAN_FSEGT (Id_book, NCE, borrow_gdate, return_date)

The local schema of the sites ENIT, FST and FSEGT are almost the same. Distribution column in horizontal fragment are removed as in  $EMPLOYE\_ENIT$  table for example. This is not considered as data loss because data location replaces each row qualification (assignment column), but storage optimization.

### C. Distribution performance evaluation

To evaluate distribution strategies, we focus on one DDB performance parameter: Execution time. We define an operation as a book subsequent borrowing and back operation. This procedure takes in charge additional checking operation as book availability and student ability to borrow (< N books at a time).

**First calculation plan**, the reference, considers execution time on a remote call to centralized database.

**Second evaluation scenario** is to fragment "STUDENT" table horizontally and it derived "LOAN" table. "BOOK" will also be split horizontally based on "UNIVERSITY" column.

**Third scenario** is built by splitting "Student" table vertically to get the "STUDENT\_BIB" fragment and then duplicate this fragment on each site.

Once implemented, we stress-test database on each scenario with 100 concurrent users for 100 borrow-return operation. The results are described in Figure 1.

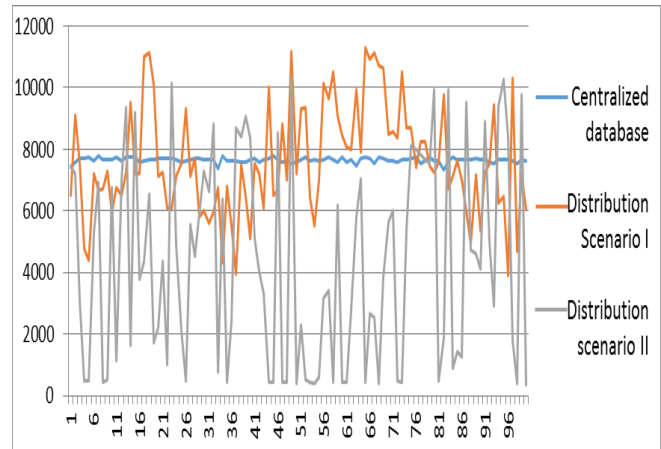


Figure 1. Load test result on each scenario

Native interpretation of the above diagram shows that first distributed scenario gives a similar or less efficient performance than centralized database. This may be explained by rising inter-sites update operations on stretched horizontal fragments. Trying to fix this issue through a nested fragmentation on "STUDENT" table made significant improvement but it is still penalizes write operations (lock acquisition duration between concurrent processes). It is worth to remembering that this evaluation omit erroneous transactions assuming that the application layer handles such constraints. Moreover, performance criteria are not the only dimension to consider in DDB evaluation [2]. Data storage optimization and transaction errors rate in the distributed context have a major impact on DDB strategy rating.

Even if all distribution scenarios seem to be valid at first sight, evaluation against worst scenario can favor some distribution scenario over others.

### D. DDB Implementation principle

DDB implementation is carried out manually. DBA must make a centralized DBMS as distributed one by granting multiple transparencies as described in [6][9]:

- i. Distribution Transparency making users ignore data replication and fragmentation. As a direct consequence, the system handles updates of all copies of a fragment.

- ii. Transaction transparency ensures global database transparency in user’s concurrent access and on system breakdown.
- iii. Performance transparency grants that the system manages efficiently queries referencing data related to multiple sites.
- iv. Queries transparency referencing data of more than one site.

DBMS transparency allows using different DBMS in the global system, without making the user aware of it.

### III. MOTIVATION

As described previously, DDBs are still facing the following issues:

- i. DDB design is not an easy task. Multiple criteria must be considered in this sensitive operation: Sites number, user needs and frequent queries.
- ii. Designer must establish a compromise between data duplication and performances cost of update and select queries. He must find out relationship to fragment, to replicate and update type to consider on each synchronous or asynchronous relationship.
- iii. Existing DDBMS have several do not have an integrated component which ensures the automatic distribution of the initial centralized database as confirmed Table V.

TABLE V. OVERVIEW OF SOME DDBMS

Prop.	RDBMS				
	ORACLE [5]	FI[8]	Cassandra [4]	Action [10]	MySql [12]
Partitioning API	Oracle Partitioning	Spanner	RP & OPP <sup>a</sup>	Ingres XOpen DTP	MySql Cluster GCE
GUI	No	No	No	No	No
Auto. Impl.	No	No	No	No	No

a. Random Partitioner & Order Preserving Partitioner

The summarized Table V shows that main market of DDBMS have partitioning APIs; but, it always in command line which request a lot of effort from designers to implement a DDB.

In the following section, we propose a new approach of DDB design and implementation assistance. This outlined approach was validated by the design and the implementation of an assistance providing designer with a graphical interface for carrying different types of database fragmentation, allocation and replication, ensuring validation on each step of the process. Once the schema has been described graphically, the system generates SQL scripts for each site of the initial described configuration (site properties). The proposed tool can be added as a layer for all existing DDBMS.

### IV. NEWAPPROACH PROPOSAL

In the section, we propose our new approach of DDB design and implementation assistance.

#### A. New Approach aims

Ideally, the new layer must satisfy the following objectives:

- i. Design help for distributed schema: The layer must provide the designer a friendly and productive interface that allows him to represent the draft of the design in to a comprehensive and accessible script to review and collaboration. Fields, tables, sites suggestion lists and work tools (fragmentation and replication) must be provided to designer to ease schema graphical description and avoid additional task complication.
- ii. Automated implementation of design schema: Once distribution schema has been established and validated by the designer along with the wizard assistance, the component “Script generator” must afford the ability to translate accurately the described distribution policy to valid SQL scripts. Generated scripts can be directly run in sites from the layer if access has already been prepared, or given deliverable files to transmit to each site administrator.

#### B. Suggested layer architecture

Figure 2 illustrates the architecture of the proposed layer.

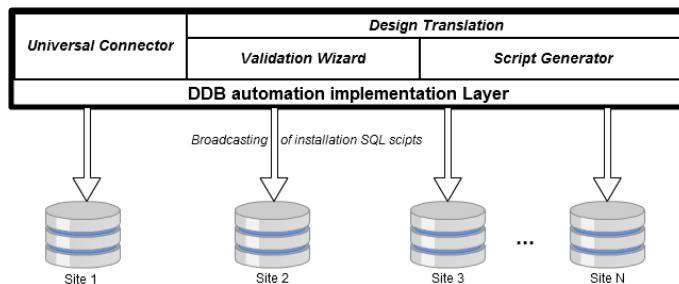


Figure 2. Layer Architecture

The implemented layer offers:

- i. Access to centralized database to distribute
  - ii. DB link creation
  - iii. Horizontal, vertical and nested fragmentation
  - iv. Fragmentation result validation
  - v. Data replication
- At the end of the process, two options are afforded to carry out scripts, depending on afforded preconditions:
- i. Automatically: If the design environment has valid access to remote sites, the layer carries out scripts in each remote site.
  - ii. Manually: User transfer files using an external tool and handles then implementation in remote sites.

#### C. Work Results Description

Oracle Database distribution wizard is intended to help users graphically distribute a centralized database, supports the creation of DB links, horizontal, vertical and hybrid derived fragmentation, validation of different types of fragmentation and replication. The end result is a set of SQL

scripts to run on each site. The given algorithm in Table VI summarizes the general functional process.

TABLE VI SUMMARY ALGORITHM OF THE ASSISTANT

```

BEGIN
accessible := FALSE; sites_count := 0; sites_list := NULL;
start_tables_list := NULL; fragments_list := NULL;
scripts_set := NULL;
WHILE (accessible = FALSE){
  read (ip, username, password);
  accessible:=check_access_to_site(ip,username,password);}
WHILE (sites_count <= 0) {  read(sites_count); }
FOR (i:=0; i < sites_count; i++){
  valid_site := FALSE; a_site := NULL;
  WHILE(valid_site = FALSE){
    a_site := create_site(read(site_info));
    IF(a_site != NULL) {valid_site := TRUE;}
    add_site(sites_list,a_site);}
start_tables_list := load_start_tables_list(ip, username, password);
IF(start_tables_list != NULL){
  finished_fragmentation := FALSE;
  WHILE(finished_fragmentation = FALSE){
    read(table_to_process); read(fragmentation_type);
    temp_fragments_list = fragment(table_to_process, fragmentation_type)
    valid_fragmentation := validate(temp_fragments_list, table_to_process,
      fragmentation_type);
    IF(valid_fragmentation.result = TRUE){
      merge_list(temp_fragments_list, fragments_list);
      show_validation_report(valid_fragmentation.report);
      read(finished_fragmentation);} }
FOR EACH(FRAGMENT f in fragments_list)
{ write("Duplicate fragment " + f.fragment_name);
  read(duplicate);
  IF(duplicate = TRUE){
    FOR EACH (Site s in sites_list){
      write(s.ip + " holds a copy of "+f.fragment_name+"?");
      read(hold_copy)
      IF(hold_copy){
        temp_frag = copy_fragment(f);
        temp_frag.site = s.ip;
        temp_frag.duplicat = TRUE;
        fragment_list.add(temp_frag);
      } } }
read(save_repository);
IF(fragments_list != NULL){
  FOR EACH (Site s IN sites_list){
    script_file_name = save_repository +
      SEPARATOR + "script_" + site+".sql";
    exists := find_file(script_file_name, scripts_set);
    if(exists = FALSE){
      creer_fichier(script_file_name);
      ecrire_lien(script_file_name,s);
      add_script(scripts_set, script_file_name);
    } FOR EACH(Fragment f IN fragments_list){
      IF(f.site = s OU f.replicat){
        transcript(f,script_file_name);
      } } }
IF(scripts_set != NULL){
  FOR EACH(FILE fc IN scripts_set){
    add_synonyms(fc);
    add_stored_proc(fc);
    add_materialized_views(fc); }
  read(auto_execute);
  IF(auto_execute){
    FOR EACH( FILE fc IN scripts_set){
      can_run:=check_access_to_site(fc);
      if(can_run){run(fc); } } }
END

```

## V. DDB HELPER

Distribution wizard "DDB Helper" is intended to help users graphically distribute a centralized DB, supports the creation of DB links, horizontal, vertical, hybrid and derived fragmentation and replication. The final result is a set of SQL scripts to run on each site.

To implement our tool, we used Microsoft Windows Seven software environment. Simulation nodes in network, was made by installing two virtual machines (Oracle Virtual Box) on the chosen host. The development environment is appeneded DotNet framework 4.5 [1].

DDB Helper provides designers with multiple screens. After welcome screen and tool introduction and interactive help access, user access the connection panel to identify target centralised database.

On successful connection test, next screen is just a popup asking for the number of sites on the distribution. Then, a visual map is displayed with raw nodes. Designer must identify each site with network address (either a name or an ip), a logical name and the DB link name.

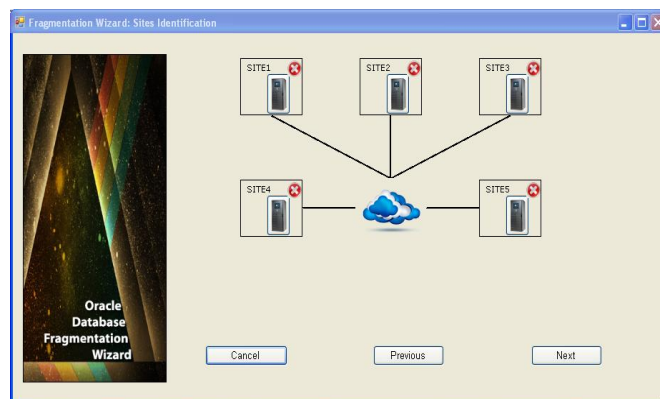


Figure 3. DDB Visual Sites Map

Next step after sites definition is the fragmentation screen. The list of accessible tables for the previously defined user is added as an auto complete on the first combobox. The second combobox suggests fragmentation types (horizontal, vertical and nested). Derived fragmentation is transparent to user. As example, vertical fragmentation interface provides user with the list of columns of chosen table. User enters fragment name and chooses hosting site and then checks columns related to this fragment. By default, the tool keeps the last selection of columns so that the designer can affect the same fragment to multiple sites without redefining then fragment columns. If the designer needs to flush selection, a shortcut on F5 key is linked and functional.

Once finished the fragmentation for a table, the wizard starts an automated validation for the described configuration. Adding a fragment without a primary key is already controlled while creating the fragment (on "Add Fragment" button click). Validation screen is displayed then: The left canvas holds a fragment tree with first level nodes as

sites, second level nodes as fragment names and leaves are the columns. Primary key is highlighted (orange color). In the right container, the validation report is displayed for the three validation criteria: Reconstruction, completeness then disjointness.

Those criteria are examined respectively with details of failure or invalid result. In the current trace we did in purpose correct fragmentation on ENIT site, then a completeness non-compliant fragmentation for site FST, and finally we forgot-in purpose- the NB\_BORROW column in two fragments to make fragmentation disjointness non-compliant. A validation trace sample is described in Table VII.

TABLE VII. SAMPLE EXECUTION TRACE OF VALIDATION PROCESS

Fragmentation result validation for table STUDENT: Reconstruction aspect test: -Fragment STUD_BIB_ENIT on ENIT has primary key. -Fragment STUD_BIB_FST on FST has primary key. -Fragment STUD_BIB_FSEG on FSEG has primary key. -Fragment STUD_ADMIN_ENIT on ENIT has primary key. -Fragment STUD_ADMIN_FST on FST has primary key. -Fragment STUD_ADMIN_FSEG on FSEG has primary key. ->Reconstruction aspect is valid on all sites. Completeness check for vertical split FST: -Completeness aspect test for site FST: ->There is no relation that can reconstruct the original table on the active distribution in site FST; Not all columns of the original table are distributed over vertical fragments. Original columns count: 8 Distinct columns count after distribution: 7 Skipping disjointness aspect test... Completeness check for vertical split ENIT: -Completeness aspect test for site ENIT: ->Completeness aspect is valid on site ENIT -Disjointness Aspect test for site ENIT ->Disjointness aspect is valid on site ENIT Completeness check for vertical split FSEG: -Completeness aspect test for site FSEG: ->Completeness aspect is valid on site FSEG -Disjointness Aspect test for site FSEG -> Detected duplicate columns(different from PK) in fragments of site FSEG: This configuration does not fill disjointness requirement. Duplicate Columns are: NB_BORROW
---

The trace shows the validation process result. Reconstruction aspect is checked first. Then, completeness aspect is checked out. In this sample, the completeness aspect is altered in site FST. When the validation wizard component tries to rebuild the parent table from its child fragments, it fails because one column is missing from all vertical fragments. Finally, disjointness check reports a broken distribution against this correctness rule because of a duplicate column different from primary key between two fragments in site FSEG. The detailed report is very helpful while stepping back to correct distribution strategy.

By the end of the whole process, if the policy is validated by the wizard and designer, the tool takes in charge the transcription of visual design into SQL scripts to run on remote sites. The only necessary parameter for this operation

is scripts location. Script files naming convention is as follows: [SITE\_NAME]\_DDB\_SCRIPT.sql.

The generation process goes through all sites and generates the script to create symbolic links, then transforms into a standard fragments and commented SQL script. The field names and types are consistent with the starting table (same name and same type). Procedures, views, triggers, and the various components are then written accordingly.

## VI. COMPARISON WITH EXISTING APPROACHES

In parallel with the work of DDBMS vendors and developers, the design of distributed Database has been investigated in many research papers. In this section, we focus on the works of Rim [11] and Hassen [7]. The first, DDB Expert: A Recommender for Distributed Databases Design proposes an open source expert system for database partitioning. The author has implemented a recommender for DB fragmentation, which infers solutions for table fragmentation using a knowledge base populated with DB schema, DB workload facts, and DB statistics.

The second is “A New Data Re-Allocation Model for Distributed Database Systems” [7]. Abdalla presents a new data re-allocation model for replicated and non-replicated constrained DDBSs by bringing about a change to data access pattern. This approach assumes that the distribution of fragments over network sites was initially performed according to a properly forecasted set of query frequency values that could be employed over sites

In our work, we help the designer to validate its fragmentation; User who chooses attribute frag. Our layer enables:

- i. Checking whether the described fragmentation is valid or not against reconstruction and completeness criteria. Disjointness is checked twice while creating fragments and on global validation. But this is a non-blocking condition because of design issues sometimes where we opt for non-empty intersection to keep inter-site relational integrity.
- ii. Automatically generate SQL scripts Materialized views definition is based on reconstitution rules of pre-established relations.
- iii. As the previous works in this field published by Rim are focused on design assistance, this work can lead to a complete distribution layer if associated with the open source work of Rim [11].

## VII. CONCLUSION AND FUTURE WORK

In this work, we have highlighted the constraints and challenges faced by designers for carrying out a DDB scheme. We have explained some concepts of DDBs and methods of design and implementation of such a database. Lack of a smart assistant that allows the automatic implementation of a database distribution policy was our starting point for the design and implementation of an assistance layer to design and implement a DDB.

The result of current work is a friendly visual wizard, which allows the translation of schemes of distributed directly on all the nodes of the topology.

Further work to improve the DDB-helper layer: 1) Full support of hard and software heterogeneity (Different DBMS, Different OS, and Network topology) and 2) integrate performance simulator (Enable designers to anticipate bottlenecks even before implementing distribution policy, predict performance interpolation graphs based on user predefined queries).

## REFERENCES

- [1] A.P. Rajshekhar, .Net Framework 4.5 Expert Programming Cookbook, Packt Publishing, 2013, pp. 45-101, ISBN: 978-1-84968-742-3.
- [2] A. Silberschatz, Distributed databases. In Database System Concepts, fifth edition, Connecticut: McGraw-Hill, pp. 705-749, 2006.
- [3] B. Pribyl and S. Feuerstein, Learning Oracle PL/SQL, O'Reilly Media, 2001, pp. 21-269, ISBN: 978-0-596-00180-3.
- [4] E. Hewitt, Cassandra: The definitive guide. O'Reilly Media, Inc., November 2010.
- [5] F. Bouzaiene, Oracle Golden Gate. In: Conférence "Oracle Technologie Day Tunis", Oradist, 27 Mars 2013, Hôtel Sheraton, Tunis.
- [6] G. QIAN, B. LIU, and J. CHEN, "Design and Implementation of Distributed Database System," Modern Surveying and Mapping, June 2010, ISSN: 1672-4097.
- [7] H.I. Abdalla, A New Data Re-Allocation Model for Distributed Database. Systems International Journal of Database Theory and Application, vol. 5, June 2012.
- [8] J. Shute and M. Oancea , S. Ellner, B. Handy, E. Rollins, S. Bart, R Vingralek, C. Whipkey, , B. Jegerlehner, K. Littlefield, T. Phoenix, F1 -The Fault-Tolerant Distributed RDBMS Supporting Google's Ad Business, 16 Mai 2012, Arizona..
- [9] M.T. Özsu and P. Valduriez, Principles of distributed database systems. New York, Springer, 2011.
- [10] M. Stonebraker, The INGRES Papers., Addison-Wesley Publishing Company, 1986.
- [11] R. Moussa, DDB Expert: A Recommender for Distributed Databases Design. Database and Expert Systems Applications (DEXA), pp. 534-538, 2011.
- [12] Y. Bassil, A Comparative Study on the Performance of the Top DBMS Systems. Journal of Computer Science & Research, vol. 1, No. 1, pp. 20-31, February 2012.

# An Object-oriented Approach for Extending MySQL into NoSQL with Enhanced Performance and Scalability

H. Shim, Y. Sohn, Y. Sung, Y. Kang, I. Kim, and O. Kwon

Samsung Electronics Co., Ltd

{hyunju67.shim, ycsohn, yw.sung, ygace.kang, ij00.kim, ohoon.kwon}@samsung.com

**Abstract**—This paper introduces an object-oriented approach for extending MySQL into Not-only SQL (NoSQL) for enhanced performance and scalability. Main goal of our system is to provide a database management system that handles a huge amount of data in a fast and reliable way. In designing database management Application Programming Interface (API), we adopted an Object-Relational Mapping (ORM) approach that provides a mapping between objects in user code with tables in database. This mapping allows developers to handle data in database tables by manipulating the objects that are mapped to. To provide a flexible and efficient distribution of large amount of data among multiple database nodes, we designed a unique ID scheme which is optimized for the primary key lookup operations by encoding the shard key information into its data ID. In this way, queries predicated with data ID can always go directly to the target node no matter which key the data is distributed on. With our ORM approach, query predicates are composed as a combination of Java method calls and no query parsing is necessary at database layer. Having no query to be parsed, we leveraged HandlerSocket, a MySQL plugin, which bypasses the upper layer of MySQL hence improving overall performance. To evaluate our system, we performed a set of tests and proved that our system provides improved performance and linear scalability compared to the traditional MySQL approach.

*Keywords*—performance, robustness, scalability, object-relational mapping, distributed query processing

## I. INTRODUCTION

Recently, due to the huge success in social networking services such as Facebook and Twitter, the amount of user data to be handled by a single service has exponentially grown. To support a service where a huge number of users generate tons of data at every second, its database management system must not only be fast and robust but also be highly scalable and available. In database system, scalability is defined as an ability to increase the total throughputs linearly as database storages are added. With highly scalable database system, simply adding more database nodes to the system will handle ever increasing user data without performance degradation. For highly available system, its database solution must eliminate any single-point-of-failure in system and guarantee its service level agreement.

Traditional database management systems (DBMS) designed to serve complex queries with Atomicity, Consistency, Isolation, and Durability (ACID) properties have demonstrated its architectural limitations in handling the large amount of data that recent social services generate. To remedy this situation,

researchers investigated in alternative database management systems that can handle large amount of data with high performance and scalability. As a result of those efforts, a new type of database management systems called NoSQL (Not only SQL) were introduced including Amazon's Dynamo [1] and Google's Bigtable [2]. Unlike data in traditional database systems were modeled and handled as relational tables, data models in NoSQL are characterized into several categories such as document-based [3], column-oriented [4], key-value pairs [5], graph-based [6], and etcetera.

Although NoSQL provides a fast access to vast amount of data with variety of data models, being recent technology, they are not robust enough compared to the traditional DBMS such as Oracle [7] and MySQL [8]. Also, being distributed database system, NoSQL has an innate trade-off between high-availability and data consistency [9] and recent trend is that developers choose NoSQL or RDBMS for specific needs depending upon the nature of their data and services [10]. In this paper, we introduce a data access framework we developed on top of MySQL to provide a fast and robust data access with high scalability and availability.

Section II describes about the overall architecture of our system. Section III and Section introduce our approach to database sharding and programming model. Section V describes our experimental results and, finally, Section VI concludes this paper and explains about our future works.

## II. OVERALL ARCHITECTURE

Main goal of our system is to provide a fast and robust data access framework with high scalability and availability as followings:

- For high scalability, we horizontally partitioned tables into multiple database nodes.
- For high availability, we automated the procedures for master failover and provided an online tool for shard rebalancing that redistributes data from one shard to another.
- For robust data management, we leveraged the MySQL storage engine, which has been used for many commercial services for a long time.
- For fast data access among multiple data nodes, we designed a unique ID scheme which is optimized for the ID lookup operation no matter which key the data is distributed on. This is done by encoding the shard key information into the data ID.



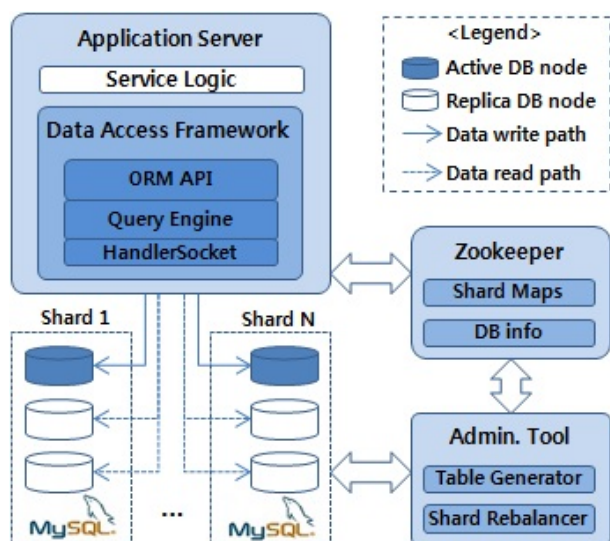


Figure 1: Overall Architecture of Our System.

- For fast data access in a single data node, we adopted HandlerSocket plug-in [11] and made read requests be routed to replica nodes.

As shown in Figure 1, our system is mainly composed of four parts: Data Access Framework in client side, which builds query conditions from user code and distributes queries among multiple MySQL nodes, Zookeeper which coordinates shard maps that resides in client nodes, MySQL database nodes with Active-Standby replications, and Administration Tools which provide command line interfaces for shard creation and online rebalancing. Among many features of our system, data distribution, programming API, performance, and scalability aspects of our system are discussed in the following sections.

### III. HORIZONTAL PARTITIONING OF DATA

#### A. Basic Concept of Database Sharding

In our system, we achieved database scalability by horizontally partitioning a single table into multiple data nodes where rows of a table are held separately based on the values of a certain key. This vertical partitioning of tables is called sharding and the key used to divide value range of data is called shard key. Two famous approaches for sharding are hash-based sharding and range-based sharding. In the hash-based sharding, data is simply distributed based on the hash values of its shard key hence data are evenly distributed among multiple data nodes. However, since data are scattered among multiple shards - data nodes, it is inefficient to perform queries with range conditions e.g., smaller than, larger than, and etcetera. For those range queries, every shard needs to be accessed to check if the node has the data that meet the range conditions. Also, with a simple hash, when a new data node is added into the system to increase the total capacity of storage, data from every node must be relocated based on the new hash function.

To remedy this data migration of every shard in a simple hash-sharding, consistent hashing was introduced [12]. In

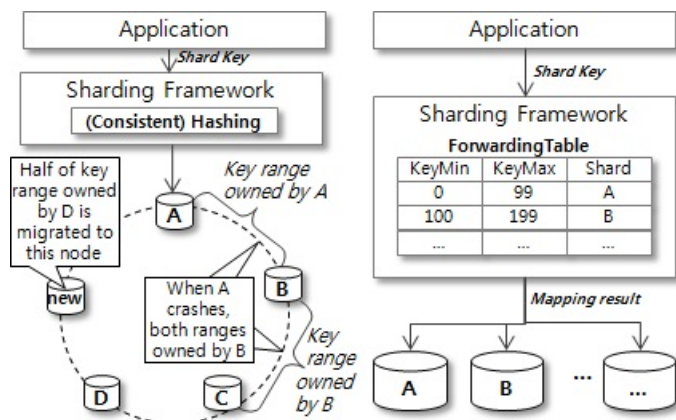


Figure 2: Concepts of Hash Sharding (left) and Rule Sharding (right).

consistent hashing, hash key space is mapped to the points on a ring representation and each point is mapped to a data node. Thus, each data node covers a certain range of hash key space represented on the ring. When the number of data nodes changes due to crashing of any node or addition of a new node, only  $K/N$  keys need to be reallocated on average where  $K$  is the number of keys and  $N$  is the number of points or data node on the ring. To reduce data migration in case of node crash, data in the nodes are replicated in such a way that each node in the ring keeps the master copy of its own key ranges and replicas of adjacent nodes. In the left image of Figure 2, data being replicated to its adjacent nodes, when the node A crashes, requests made to A are then routed to node B hence node B became the master for the data whose hash key space is mapped to A and B. When a new node is added, a hash key range owned by D is divided into two and half of them are migrated to the newly added node letting the newly added node become a new master node of the migrated half.

Unlike hash-based sharding, rule-based sharding groups shard key ranges and maps the groups to a set of data nodes. The file that keeps this mapping information is normally referred to as a forwarding table. Advantages of rule-based sharding are flexible data distribution and efficient support for ranged-queries. With range-based sharding, database administrators can make sharding rules specific to their applications and data with similar shard key values are stored in the same data node hence supporting range queries efficiently. One of main disadvantages of range-based sharding is uneven distribution of data. If the sharding rule does not reflect the actual distribution of data then some of data nodes might get congested while others are sparse. To make data nodes evenly distributed, frequent data migrations and shard rule updates might be required. Right image of Figure 2 illustrates the main concept of rule-based sharding. In our system, we designed a modified rule-based sharding with doubly mapped forwarding table and fixed length of ID. Following section explains details about the sharding in our system.

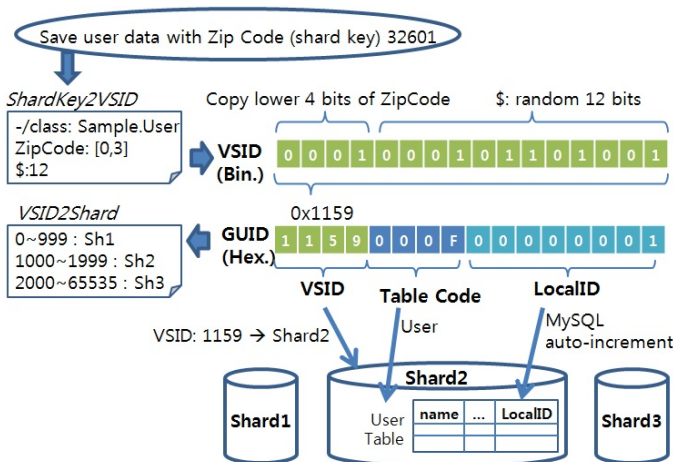


Figure 3: How Sharding Works in Our System.

### B. Doubly Mapped Range-based Sharding

When distributing data among multiple nodes, they must be grouped based on the values of a certain key, which is called a shard key. For efficient distribution of data, it is important to select a shard key that is common for most of queries in the system. Because, once data is distributed among multiple shards based on their shard key values, queries predicated with a shard key can be directly sent to the target shard by referring to the forwarding tables while others must be sent to every shards to see if the data to be queried reside in there.

For an efficient sharding of data where most of their queries are ID lookup operations, we designed a 64 bits ID scheme in such a way that the shard key values are encoded within it. With this approach, data is distributed based on any key while the ID lookup operations can always go directly to its destination shard by referring to the shard rule files. In the ID scheme we designed, the first 16 bits represent the virtual shard ID (VSID), which is mapped to a physical database node, the middle 16 bits represent a database table and the remaining 32 bits represent a local ID which is unique within the table.

For data read and write operations, our system maintains two mapping files: *ShardKey2VSID* and *VSID2Shard* the *ShardKey2VSID* file maps the shard key value ranges to  $2^{16}$  VSIDs and the *VSID2Shard* file maps the  $2^{16}$  VSIDs to the physical database nodes in system. For data writing, a VSID is first assigned based on its shard key value by referring to the *ShardKey2VSID* rule file. Once a VSID is assigned, a target database is determined by referring to the *VSID2Shard* rule file. Once the physical shard determined, data is stored in the data types table with the local ID provided by the MySQLs auto-increment functionality. Figure 3 shows how ID is generated for data and stored in our system. For data reading with ID, our system looks at the first 16 bits of the ID, which is the VSID part, and finds the target shard by referring to the *VSID2Shard* file. For data queries including shard keys values can also be directly sent to the destination shards by comparing the shard key values with the *ShardKey2VSID*

and *VSID2Shard* files. Like other sharding systems, queries without shardkeys must be sent to every shard.

One of the main design goals of our VSID is to support flexible sharding of data with multiple shard keys while encoding the shard key values into the fixed length of ID. Using our system, a database administrator can set rules for each 16 bits to distribute data according to the characteristics of their applications. For example, sharding a user data according to the birthday and zip code in such a way that the upper 8 bits of VSID are set based on the birthday and the lower 8 bits are set based on the zip code of users will result that user data with similar birthday and zip code will be stored in the same database. If related data are gathered in a single shard, operations such as transactions and joins can be performed locally using SQL in user code hence providing better performance. Note that transaction and join operations among distributed data are very complex and time consuming tasks. Also note that, being Java API, users can use our API and standard SQL statements together in their code for simple data look up operations and transaction and join operations, respectively.

## IV. PROGRAMMING MODEL

In relational database systems, the Structured Query Language (SQL) provides a standard way to access and manipulate data in database tables. To access data in RDBM tables from an application, developers normally compose SQL statements in a string representation and request/execute the query using the database client API. Figure 4 shows the example code for creating a database table and inserting/querying data into/from the table using SQL in Java.

Although SQL provides a standard and structured way to store and retrieve data in tables, when used in Object-Oriented (OO) programming languages where data to be manipulated is represented as properties of object, there exists a gap between the representations of data in the programming code side and in the database side. That is, data in database tables are represented as a set of columns with scalar values while the data in OO programs are represented as properties with associated get/set methods. While developers can manually map the objects in their code into the rows of tables, some Object-Relation Mapping (ORM) frameworks have been introduced to free developers from this manual mapping [13][14]. Using ORM API developers can save and retrieve objects into and from the relational database tables as if they are manipulating their objects using standard OO methods. In our system, we designed and developed a set of ORM API which is highly optimized for building and requesting complex queries to database tables which are physically distributed in a fast way. Following sections describe details about accessing and manipulating data in tables from Java code using our ORM API set.

### A. Data Definition

While relational database tables are normally created using the Data Definition Language (DDL) statements of SQL as shown in Figure 4, with the ORM approaches, tables can also

```

//Create Table
conn = DriverManager.getConnection();
st = conn.createStatement();
String sql = "CREATE TABLE User (
    U_Id int NOT NULL, name varchar(8) NOT NULL,
    email varchar(8), addr varchar(8),
    PRIMARY KEY (U_Id),
    FOREIGN KEY (P_Id) REFERENCES Persons(P_Id)) ";
st.execute(sql);

//Create Row
connection = DriverManager.getConnection();
preparedStatement =
prepareStatement(connection, "INSERT INTO Authors
(firstname, lastname) VALUES (?, ?)");
int affectedRows = preparedStatement.executeUpdate();

//Query
connection = DriverManager.getConnection();
preparedStatement = prepareStatement(connection,
"SELECT id, email FROM User WHERE name=? ", false,values);
resultSet = preparedStatement.executeQuery();
if (resultSet.next()) { user = map(resultSet);
    
```

Figure 4: Example Code of SQL Statement in Java Code.

```

Session s = Loom.newsession();

//Insert a user whose name is John and 40 years old
User newUser = new user();
newUser.setName("John");
newUser.setAge(40);
s.save(newUser);

//Update users' age to 32 whose name is "Jane"
User jane = s.build(UserQuery.class).named("Jane").fetch();
jane.setAge(32);
s.save(jane);

// Delete users whose name is Tom
User tom = s.build(UserQuery.class).named("Tom").fetch();
s.delete(tom);
s.commit();

//Query user whose name is "John" and age is older than 19
UserQuery q =
s.build(UserQuery.class).olderThan(19).named("John");
List<User> users = q.fetchlist();
    
```

Figure 6: Composing a Query and Fetching Data.

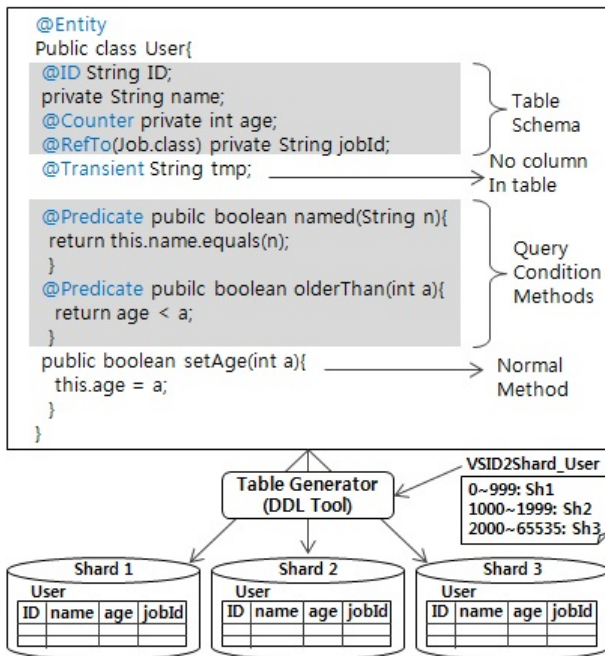


Figure 5: Data Definition and Query Condition in Our System.

be created from the annotations in the source code or from XML documents that specify mapping between the objects and database tables [13][14]. In our system, for the definition of the structure and schema of database tables across the distributed database tables, we first designed a set of Java annotations and then developed a tool that reads the annotations in a Java code and creates the tables in the sharded databases according to

the pre-defined sharding rules.

Using our system, developers can create tables with a primary key, a foreign key, an auto-increment column using our annotations such as `@Entity`, `@ID`, `@RefTo`, and `@Counter` annotations, respectively. For every class with `@Entity` annotation, a corresponding table is created in database nodes. Class variables with no annotations will be created as a simple column with its data type defined in the code. For the class variables with `@Transient` annotation, no matching column will be created in the table. Figure 5 shows the example code for the creation of a table in our system and basic concepts about how the tables are created in multiple shards by the DDL tool of our system. In Figure 5, the `User` table is created in three shards according to the rules in `VSID2Shard` file.

We also provide a predicate annotation for methods where developers can specify query predicates to be made for the correspondent table. Middle part of the Figure 5 shows the example `@Predicate` methods which compare the method parameters with the values of the name and age of the entity object and return the comparison results. This `@Predicate` annotated method is a building block for complex queries to the database table that corresponds to the current entity object. In the application code, developers can compose a complex query to a database table by calling the `@Predicate` annotated methods of the corresponding entity object. Following section explains how to build a complex query by leveraging this `@Predicate` annotated methods in detail.

### B. Data Manipulation

With our ORM approach, developers can manage data in a table as they manage them with object. To insert data into a table, developers simply create an object with values to its member variables and provide the object as a parameter of our `save()` API. To update data in a table, developers first fetch the row of the table to be updated using our `fetch()` API

note that our *fetch()* API fetches data from database tables and maps them to the object upon which the data request is made. Once the row is fetched as an object, update can be made to the object using the *set()* method and save the object into the database table again using our *save()* API.

In our system, complex queries can be built by combining the *@Predicate* annotated methods that were pre-defined in the *@Entity* class. For example, using the predicate methods in Figure 5, the *named* and *olderThan* methods, we can build a query for selecting users whose name is John and whose age is older than 19 as in the last example in Figure 6. Note that composition of a query is achieved by sequentially calling the *@Predicate* methods of the Entity class. Once the query condition is built, developers can fetch data that meet the query conditions by calling the *fetch()* or *fetchlist()* API at the end of the query conditions built. For the *fetch* or *fetchlist()* methods, our query engine takes the user-built query and fetches the data that meet the conditions from the multiple shards.

Advantages of our ORM approach can be summarized in two points. First one is that since class variables and methods are mapped to a database tables and query predicates, application developers can manage data in database transparently - that is the application developers even do not need to aware the database behind the application and manage data as if they are managing objects in their application. This satisfies the goal of ORM, which is eliminating the conceptual gap between the objects in OO programs and the relational tables. Also, since there is no string manipulation for composition SQL statements, many typo errors can also be eliminated.

Second advantage of our approach is that since the query conditions are defined as class methods, developers can specify query conditions using the operators allowed in SQL WHERE clause such as LIKE, IN, MAX, etc., along with the language operators, such as bitwise operators and variety of string operators. With SQL approach, developers apply additional language operators to the SQL query results to meet the certain requirements of the application. In our system, the query condition defined using language operator is called a client query and the query condition that can be mapped to the SQL WHERE clause is called a DB query. Next section describes details about the query engine of our system.

### C. Query Engine Internals

Query engine in our system is responsible for constructing a database query from the query predicates in application code and making request to database tables in multiple shards and, finally, returning the query results to the application as an object. Figure 7 illustrates the internals of our query engine. Once the *build()* API is called, our query engine starts building an empty query object that correspondent to the query class specified in its argument. In Figure 6, this correspondence to *s.build(UserQuery.class)*. Once an empty query object is ready, the query engine adds the query conditions to the query object as it is specified in the application. In Figure 6, this correspondence to *olderThan(19)* and *named(John)*.

Once the query object is built with the given query conditions, our query engine separates the database query conditions

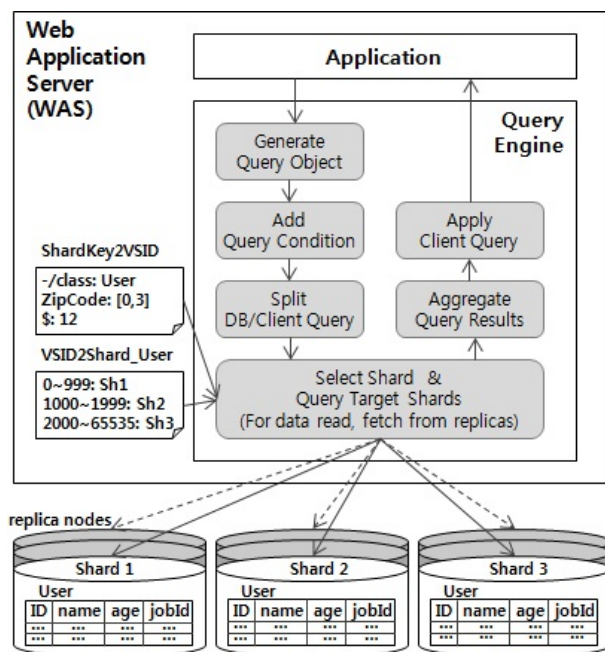


Figure 7: Internals of Query Engine.

from the client query conditions. Previously, it was mentioned that, the query condition defined using language operator is called a client query and the query condition that can be mapped to the SQL WHERE clause is called a DB query. As the database queries are filtered out, our query engine converts the database queries into a format of HandlerSocket protocol. Finally, the query engine refers to the sharding rule files to identify the target shards where the query must be sent and makes database request to them.

As the database query results are sent back from the target shards, the query engine aggregates the results and applies the previously filtered-out client query conditions to the results and, finally, returns the result to the application as an object. Note that, at the bottom of Figure 6, the result of *fetchlist()* is directly assigned to the list of User objects.

## V. PERFORMANCE AND SCALABILITY

### A. Performance Enhancement

To process SQL statements in the server-side, MySQL upper layer includes a parser and optimizer that builds a parse tree and makes a plan for an optimized execution path for the given query, respectively. Although this parser and optimizer handles complex SQL queries in an efficient way, they can also be a burden when the query is very simple so that there is not much thing to do for parsing and optimizing. For fast data access with simple query conditions such as ID lookups and index searches, HandlerSocket [11] was introduced which bypasses the upper layer of MySQL hence improving read and write performances. In our system, we implemented our ORM API using HandlerSocket protocols that communicate

with HandlerSocket plug-in in server-side and directly access data in MySQL's storage engine, InnoDB.

One more performance enhancement we made to our system is leveraging replica nodes to serve for read requests. MySQL provides a built-in master-slave replication where read/write operations go to the master node while the data updates are copied to the replica nodes asynchronously [15]. Although the main purpose of this replication is keeping extra copies of data for high availability, we can also achieve an improved performance by routing read requests to one of replicas and reducing bottleneck in the master node. In our system, we designed our query engine to read data from replica nodes in a round-robin fashion. If a certain replica node is not responding then one of the other replica nodes in the same shard is read and the failed replica node is tried after some time.

### B. Scalability Enhancement

Earlier in this paper it was mentioned that, linear scalability of database is defined as an ability to increase the total throughput linearly as database nodes are added. Being a range-based sharding solution, adding a new data node and redistributing data in our system is achieved in following steps by our shard rebalancing tool:

- 1) Data migration - migrate the data of a shard whose volume reaches its threshold. This involves copying data from a congested node to the newly added node and deleting the copied data from the congested node. This way, the congested node and the newly added node shares data load.
- 2) Sharding rule update - Update the sharding rule files accordingly and synchronize the updated rule files in every WAS node so that the requests for the migrated data go to the newly added node.

Figure 8 illustrates the concept of the shard rebalancing in our system. In our system, Apache Zookeeper [16] is used for the automatic synchronization of the mapping rule files reside in every WAS nodes.

### C. Evaluation Results

To measure the performance and scalability of our system, we performed a set of tests. Firstly, to compare the performance of our approach with the traditional MySQL approach, we wrote two sets of test codes that access tables in database - one using JDBC API and one using our ORM API. Then we set up 1 database node and increased the total number of application nodes to measure the saturated throughput per second (TPS) and average latency for two different approaches. Note that we made each application node spawn 100 threads that run our test codes and to get the saturated throughput for 1 database node we should increase the total number of application nodes up to 28. The test results demonstrated that our system outperforms the MySQL approach for 34 and 13 times in terms of throughput and saves 68% and 74% in terms of latency for read and write operations, respectively. Detailed test results are summarized in Table I and Figure 9.

To measure the scalability of our system, we increased the total number of database nodes from 1 to 8 and measured

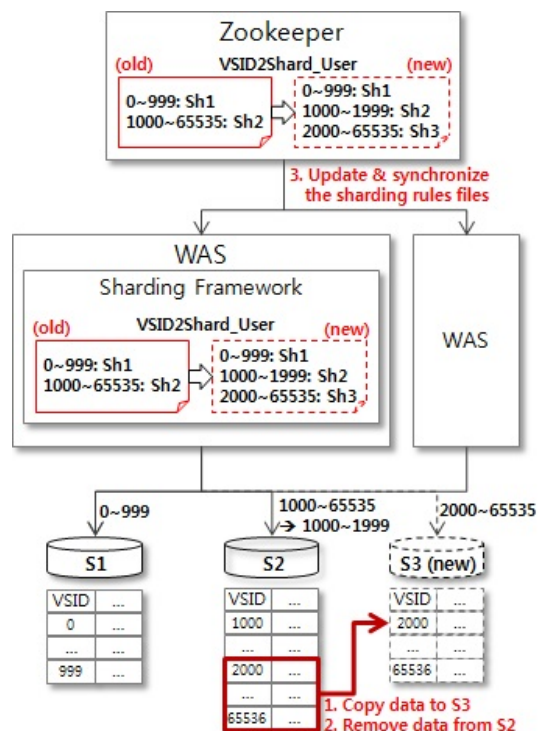


Figure 8: Shard Rebalancing.

TABLE I: Result of Performance Test

	Throughput		Latency (ms)	
	SQL-JDBC	ORM-HS	SQL-JDBC	ORM-HS
Data Read	5365	179922	2.0	0.64
Data Write	3902	49627	2.9	0.74

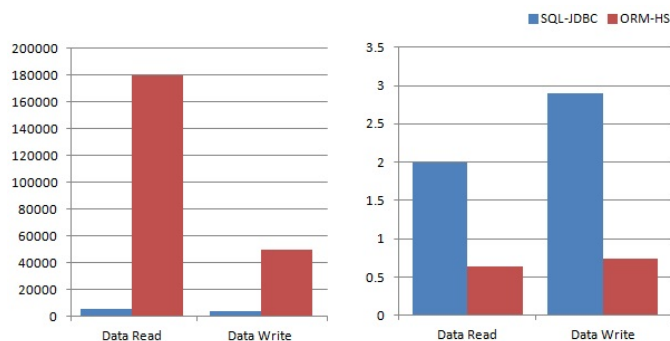


Figure 9: Result Graphs for Throughput Test(Left) and Latency Test(Right).

the aggregated TPS of all 28 application nodes. As a result of this test, we proved that our system linearly scales out as the number of database nodes gets increased. Table II and Figure 10 show our test results in detail.

TABLE II: Result of Scalability Test

	1 DB node	4 DB nodes	8 DB nodes
Data Read	179922	579368	843768
Data Write	49627	169031	303800

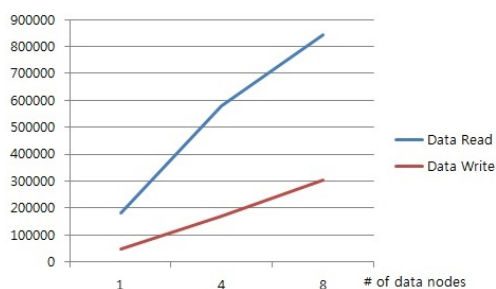


Figure 10: Result Graph for Scalability Test.

## VI. CONCLUSION AND FUTURE WORK

In this paper, we introduced our database sharding framework with enhanced performance and scalability. For a flexible and efficient distribution of data among multiple database nodes, we designed a unique ID scheme which is optimized for the primary key lookup operations by encoding the target database node information as a part of ID. We also designed a set of ORM API that maps the conditioned objects in application codes to the database table and queries. For fast data access, we implemented our API using MySQL HandlerSocket plug-in which bypasses the upper layer of MySQL. We also leveraged replica nodes to serve for read requests for an enhanced performance by distributing requests to a master node.

To evaluate performance and scalability of our system, we performed a set of tests and proved that our system provides improved performance and linear scalability compared to the traditional MySQL solutions. As our future work, we firstly plan to provide zero-downtime availability in case of a master node failure by implementing an active-active replication with the quorum based algorithm [17]. We also plan to support transactions and map-reduce style queries such as order-by and join in the distributed environment.

## REFERENCES

- [1] G. DeCandia et al., "Dynamo: Amazons Highly Available Key-value Store," In Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles, ACM New York, NY, USA, Oct. 2007, pp. 205-220
- [2] F. Chang et al., "Bigtable: A distributed storage system for structured data," ACM Transactions on Computer Systems (TOCS), ACM New York, NY, Jun. 2008, pp. 1-26.
- [3] K. Banker, 2011, MongoDB in Action, Manning Publications
- [4] E. Hewitt, Cassandra: The Definitive Guide, O'Reilly Media Inc.
- [5] T. Macedo and F. Oliveira, Redis Cookbook: Practical Techniques for Fast Data Manipulation, O'Reilly Media, 2011.
- [6] Neo4j, the Graph Database - Learn, Develop, Participate. [Online]. Available: <http://www.neo4j.org/> [retrieved: 02, 2014]

- [7] K. Loney, Oracle Database 11g The Complete Reference, Oracle Press, 2009.
- [8] B. Schwartz, P. Zaitsev, and V. Tkachenko, High Performance MySQL: Optimization, Backups, and Replication, O'Reilly Media, November 2011.
- [9] W. Vogels, All Things Distributed: Eventually Consistent, [Online]. Available: [http://www.allthingsdistributed.com/2007/12/eventually\\_consistent.html/](http://www.allthingsdistributed.com/2007/12/eventually_consistent.html/) [retrieved: 02, 2014]
- [10] N. Leavitt, "Will NoSQL Databases Live Up to Their Promise?," Computer, volume 43, number 2, Feb. 2010, pp. 12-14.
- [11] HandlerSocket. [Online]. Available: <http://yoshinorimatsunobu.blogspot.kr/search/label/handlersocket/> [retrieved: 02 2014]
- [12] D. Karger et al., "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web," In ACM Symposium on Theory of Computing, Philadelphia, Pennsylvania, USA, May. 1997. pp. 654-663.
- [13] Hibernate JBoss Community. [Online]. Available: <http://www.hibernate.org/> [retrieved: 02, 2014]
- [14] Java Persistent API. [Online]. Available: <http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html/> [retrieved: 02, 2014]
- [15] S. Pachev, Understanding MySQL Internals, O'Reilly Media, 2007.
- [16] Apache Zookeeper. [Online]. Available: <http://zookeeper.apache.org/> [retrieved: 02, 2014]
- [17] D. Agrawal and A. E. Abbadi, The tree quorum protocol: an efficient approach for managing replicated data, in Proceedings of the sixteenth international conference on Very large databases, Brisbane, Australia, Sep. 1990, pp.243-254.

# Efficient Data Integrity Checking for Untrusted Database Systems

Anderson Luiz Silvério  
and Ricardo Felipe Custódio

Laboratório de Segurança em Computação  
Universidade Federal de Santa Catarina  
Florianópolis, Brazil  
Email: anderson.luiz@inf.ufsc.br  
custodio@inf.ufsc.br

Marcelo Carlomagno Carlos

Information Security Group  
Royal Holloway University of London  
Egham, Surrey, TW20 0EX, UK  
Email: marcelo.carlos.2009  
@rhul.ac.uk

Ronaldo dos Santos Mello

Grupo de Banco de Dados  
Universidade Federal de Santa Catarina  
Florianópolis, Brazil  
Email: ronaldo@inf.ufsc.br

**Abstract**—Unauthorized changes on database contents can result in significant losses for organizations and individuals. This brings the need for mechanisms capable of assuring the integrity of stored data. Existing solutions either make use of costly cryptographic functions, with great impact on performance, or require the modification of the database engine. Modifying the database engine may be infeasible in real world environments, especially for systems already deployed. In this paper, we propose a technique that uses low cost cryptographic functions and is independent of the database engine. Our approach allows for the detection of malicious data update, insertion and deletion operations. This is achieved by the insertion of a small amount of protection data in the database. The protection data is calculated by the data owner using Message Authentication Codes. In addition, our experiments have shown that the overhead of calculating and storing the protection data is lower than previous work.

**Keywords**—Data Integrity; Outsourced Data; Untrusted Database; Data Security.

## I. INTRODUCTION

Database security has been studied extensively by both the database and cryptographic communities. In recent years, some schemes have been proposed to check the integrity of the data, that is, to check if the data has not been modified, inserted or deleted by an unauthorised user or process. These schemas often try to solve one of the following aspects of the data [1], [2]:

- *Correctness*: From the viewpoint of data integrity, correctness means that the data has not been tampered with.
- *Completeness*: When a client poses a query to the database server it is returned a set of tuples that satisfies the query. The completeness aspect of the integrity means that all tuples that satisfy the posed query are returned by the server.

Trying to assure data integrity, many techniques have been proposed [3], [4], [5], [6]. However, most of them rely on techniques that require modification of the database kernel or the development of new database management systems. Such requirements make the utilization of the integrity assurance mechanisms in real-world scenarios difficult. This effort becomes more evident when we consider adding integrity protection to already deployed database systems.

Most of the remaining work uses authenticated structures [7], [8], [9], based on Merkle Hash Trees (MHT) [10] or Skip-Lists [11]. These works are most simpler to put in practice, since they don't require modifications to the kernel of the Database Management System (DBMS). However, the use of authenticated structures limits its use to static databases. Authenticated structures are not efficient in dynamic databases because for each update the structure must be recalculated.

In this paper, we address the problem of ensuring data integrity and authenticity in outsourced database scenarios. Moreover, we provide efficient and secure means of ensuring data integrity and authenticity while incurring minimal computational overhead. We provide techniques based on Message Authentication Codes (MACs) to detect malicious and/or unauthorized insertions, updates and deletions of data. In this paper, we extend the work of [12], by enhancing the experimental evaluation, providing the algorithms for the proposed techniques and presenting a technique to provide *completeness* assurance of queries.

The remainder of this paper is divided into five sections. In Section II, we discuss related work. In Section III, we present techniques for providing data integrity and authenticity assurance. In Section IV, we analyse the performance impact of our proposed method and Section V presents our final considerations and future works.

## II. RELATED WORK

The major part of integrity verification found in literature is based on authenticated structures. Namely, Merkle Hash Trees MHT [10] and Skip-Lists [11].

Li et al. [4] present the Merkle B-Tree (MB-Tree), where the  $B^+$ -tree of a relational table is extended with digest information as in an MHT. The MB-Tree is then used to provide proofs of correctness and completeness for posed queries to the server. Despite presenting an interesting idea and showing good results in their experiments, their approach suffers from a major drawback. To deploy this approach, the database server needs to be adapted as the  $B^+$ -tree needs to be extended to support an MHT. Such modifications may not

be feasible in real world environments, especially those that are already in use.

Di Battista and Palazzi [7] propose to implement an authenticated skip list into a relational table. They create a new table, called *security table*, which stores an authenticated skip list. The new table is then used to provide assurance of the authenticity and completeness of posed queries. This approach overcomes the requirement of a new DBMS, present in the previous approach. While only a new table is necessary within this approach, its implementation can be done as a plug-in to the DBMS. However, the experimental results are superficial. It is not clear what is the actual overhead in terms of each SQL operation. Moreover, their experiments show that the overhead increase as the database increases, while in our approach the overhead is constant in terms of the database size.

Miklau and Suciu [9] implement a hash tree into a relational table, providing integrity checks for the data owner. The data owner needs to securely store the root node of the tree. To verify the integrity, the clients need to rebuild the tree and compare the root node calculated and stored. If they match, the data was not tampered with. Despite using simple cryptographic functions, such as hash, the use of trees compromises the efficiency of their method. A tuple insert using their method is 10 times slower than a normal insert, while a query is executed 6 times slower. In our experiments, presented in section IV, we show that the naive implementation of our method is as good as their method.

E. Mykletun et al. [13] study the problem of providing *correctness* assurance of the data. Their work is most closely related to what we present in this paper. They present an approach for verifying data integrity, based on digital signatures. The client has a key pair and uses its private key to sign each tuple he/she sends to the server. When retrieving a tuple, the client uses the correspondent public key to verify the integrity of the retrieved tuple. This work was extended by Narasimha and Tsudik [14] to also provide proof of *completeness*.

The motivation of the authors to use digital signature is to allow integrity checking in multi-querier and multi-owner models. Therefore, for multi-querier and multi-owner models, their work is preferable. On the other hand, if the querier and the data owner are the same, our work can provide integrity assurance more efficiently. Moreover, our method can provide the same security level while consuming less of the servers resources. This is possible because to achieve the same security level, asymmetric keys are larger than symmetric keys. For example, for achieving the security level of a 2048 bit long asymmetric key, we need a 112 bit long symmetric key [15]. This reduces the amount of data required to control the integrity by a factor of 18, meaning that the data owner will be able to outsource more data.

Additionally, following a different approach, Xie et al. [16] proposes a probabilistic method to audit queries of outsourced databases. They insert a small quantity of fake tuples along with the real tuples of the database to control and audit the integrity of the system. Their method shows to be efficient, since it doesn't require any complex functions. However, their

focus is on query integrity while in our work we are focused on the integrity of the data itself.

### III. PROVIDING INTEGRITY ASSURANCE FOR DATABASE CONTENT

To achieve a low cost method to provide integrity and authenticity, we propose to perform the cryptographic operations on the client side (application), using of Message Authentication Codes (MAC) [17], [18]. The implementation consists of adding a new column to each table. This new column stores the output of the MAC function applied to the concatenation (||) of the attributes (all columns, or a subset of them) of a row  $n$ , as shown in (1). The function also utilises a key  $k$ , which is only known by the application. The value of the MAC column is later used to verify integrity and authenticity.

$$MAC_n = MAC(k, Column1||...||Columni) \quad (1)$$

The use of a MAC function ensures the integrity of the INSERT and UPDATE operations. However, the table is still vulnerable to the unauthorized deletion of rows. To overcome this issue, we propose a new algorithm for linking sequential rows, called "Chained-MAC (CMAC)". The result of the CMAC is then stored into a new column. The value of this column, given a row  $n$ , a key  $k$ , and  $MAC_n$  as the MAC value of the row  $n$ , is calculated as shown in (2), where  $\oplus$  denotes the exclusive OR operation (XOR).

$$CMAC_n = MAC(k, (MAC_{n-1} \oplus MAC_n)) \quad (2)$$

The use of CMAC provides an interesting property to the data stored in the table where it is used. When used, the CMAC links the rows in a way that an attacker cannot delete a row without being detected, since he does not have access to the secret key to produce a valid value to update the CMAC column of adjacent rows. Moreover, calculating the CMAC is very efficient, since we calculate only two MACs and a  $\oplus$ . Updating rows is also efficient. The CMAC is not a cascading operation, that is, it only needs to be updated when the MAC of a given row is updated. Figure 1 shown an example of a table with the MAC and CMAC columns. The circles represent the value of the MAC/CMAC and the arrows shows the MACs used to calculate a specific CMAC.

Despite linking adjacent rows, any subset of the first and last rows can be deleted without being detected. This is possible because the first row has no previous row and the last row does not have a subsequent row to be linked with. To overcome this issue, we propose changing the CMAC to a circular method. That is, for the first row, the  $n-1$ -th row to be considered will be the  $n$ -th row (i.e. the last row). With this change, if the last row is deleted, the integrity check will fail for the first row. Similarly, since the first row now has a predecessor, integrity checks can start at the first row (in the regular mode it would always start in the second row).

It is important to notice that the introduction of the CMAC brings a new requirement: the table must be ordered by some



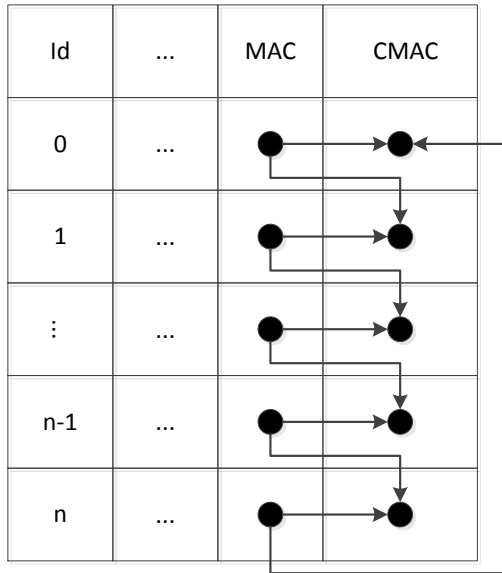


Fig. 1. Graphical representation of a table with the CMAC column

attribute. However, in real world scenarios, all tables have a primary key, and all the main DBMS orders the tables in terms of the primary key. Therefore, the requirement for a ordered table of the CMAC does not have a big impact to the deployment of our technique in real world scenarios.

#### A. Adding rows

The insertion of a new row into the database is straightforward. The client must calculate the MAC as shown in (1). The value of the CMAC column must be calculated using the MAC value of the previous row and the MAC value of the row being added, as shown in (2). Figure 2 shows the algorithm for the insertion of both MAC and CMAC values.

**Input:** A set  $T$  of values  $t_0, \dots, t_i$ , following the schema of a table  $R$  and a key  $k$ , used to calculate the MAC.

**Step 1:** Calculate  $MAC_{n+1} = MAC(k, t_0 || \dots || t_i)$

**Step 2:** Calculate the CMAC

**Step 2.1:** Retrieve the MAC of the last row of  $R$ , denoted by  $MAC_n$

**Step 2.2:** Calculate  $CMAC_{n+1} = MAC(k, (MAC_n \oplus MAC_{n+1}))$

**Step 3:** Recalculate the CMAC of the first row (for the circular CMAC)

**Step 3.1:** Retrieve the MAC of the first row of  $R$ , denoted by  $MAC_1$

**Step 3.2:** Calculate  $CMAC_1 = MAC(k, (MAC_{n+1} \oplus MAC_1))$

**Step 4:** Insert the set  $T$  along with the calculated  $MAC_{n+1}$ ,  $CMAC_{n+1}$  and  $CMAC_1$

Fig. 2. Algorithm for inserting a new row with the MAC and CMAC values

#### B. Updating rows

Updating rows is similar to the INSERT operation. The MAC and CMAC columns must be recalculated with the updated values. However, an extra step is necessary, which is to update the CMAC value of the next row, since the MAC of its previous row has been updated. Figure 3 shows the algorithm for updating a row.

**Input:** A set  $T$  of values  $t_0, \dots, t_i$ , following the schema of a table  $R$  and a key  $k$ , used to calculate the MAC.

**Step 1:** Calculate  $MAC_n = MAC(k, t_0 || \dots || t_i)$

**Step 2:** Calculate the CMAC

**Step 2.1:** Retrieve the MAC of the previous row of  $R$ , denoted by  $MAC_{n-1}$

**Step 2.2:** Calculate  $CMAC_n = MAC(k, (MAC_{n-1} \oplus MAC_n))$

**Step 3:** Update the set  $T$  along with the calculated  $MAC_n$  and  $CMAC_n$

**Step 4:** Calculate the CMAC of the  $n + 1$ th-row. If the  $n$ th-row is the last row in the table, then the  $n + 1$ th-row to be considered will be the first row of the table.

**Step 4.1:** Retrieve the MAC of the  $n + 1$ th-row row of  $R$ , denoted by  $MAC_{n+1}$

**Step 4.2:** Calculate  $CMAC_{n+1} = MAC(k, (MAC_n \oplus MAC_{n+1}))$

**Step 4.3:** Update the calculated  $CMAC_{n+1}$

Fig. 3. Algorithm for updating a row with the MAC and CMAC values

#### C. Deleting rows

To delete a row of a table that uses only the MAC column, no additional actions are needed. The reason for this is that, by using only MAC, it is not possible to check the integrity of a table against unauthorised row deletion. When using a CMAC column, the application needs to recalculate the value of the next row by referencing the previous row. Figure 4 shows the algorithm for deleting a row.

**Input:** A set  $T$  of values  $t_0, \dots, t_i$ , following the schema of a table  $R$  and a key  $k$ , used to calculate the MAC.

**Step 1:** Delete  $T$

**Step 2:** Calculate the CMAC of the  $n + 1$ -th row. If the  $n$ -th row is the last row, then the  $n + 1$ -th row to be considered is the first row of  $R$ .

**Step 2.1:** Retrieve the MAC of the  $n + 1$ -th row of  $R$ ,  $MAC_{n+1}$  and the MAC of the previous row,  $MAC_{n-1}$

**Step 2.2:** Calculate  $CMAC_{n+1} = MAC(k, (MAC_{n-1} \oplus MAC_{n+1}))$

**Step 2.3:** Update the calculated MAC,  $CMAC_{n+1}$

Fig. 4. Algorithm for deleting a row with the MAC and CMAC values

#### D. Verifying the integrity of a table

Data integrity can be provided in different levels of granularity. We can perform integrity checks of a table (entire

relation), a row (a record or a tuple of the table) or a column (an attribute of the relation). Providing integrity checks at the table level, implies that every row of the table must have the MAC column filled and its value must be calculated based on every attribute of the row. Providing integrity checks of a single row implies that that specific row must have the MAC column filled with the result of the MAC function applied on the concatenation of all attributes. Finally, providing integrity checks at the attribute level implies that the MAC function must be applied to a specific set of attributes only.

To verify the integrity of a row with the MAC column, the application must calculate the MAC of that row and compare it with the value of the MAC column. The row can be considered as not modified if the calculated MAC is equal to the stored MAC. Applying this comparison to each row of a table will ensure the integrity of this table against insertion and modification attacks. As stated earlier, the use of the MAC does not provide a means to verify the integrity of a table against unauthorized deletions. In this case, the CMAC column should be used. To verify the integrity of a table with the CMAC column, the application must check the integrity of each pair of sequential registries of the table. That is, a Table  $T$  has not been (unauthorized) modified if:

$$\forall t_{n-1}, t_n \in T : t_n.CMAC = CMAC(k, t_{n-1}, t_n) \quad (3)$$

#### E. Verifying the completeness of queries

The CMAC mechanism can also be used for verifying the *completeness* of simple range queries. This is possible due to the concatenation of adjacent rows. If the data owner does not trust the DBMS software on the server and therefore needs a guarantee that the server is not omitting valid results for posed queries, the client should proceed as shown in Figure 5.

**Input:** A query  $Q$

**Step 1:** Pose  $Q$  to the server, which will return a set  $T$  of values  $t_i, \dots, t_j$ , where  $i \leq j$

**Step 2:** Retrieve the rows  $t_{i-1}$  and  $t_{j+1}$

**Step 3:** Verify the integrity of the values  $t_{i-1}, t_i, \dots, t_j, t_{j+1}$ , as described in section III-D

Fig. 5. Algorithm for verifying the *completeness* of a query

If the result of the integrity check is positive, then we know that no intermediate result has been omitted by the server. However, just verifying the integrity does not guarantee that a value has not been omitted at all. If the server omits the values in the edge, the integrity check will still pass. To guarantee that the values in the edge have not been omitted, the client needs to check whether the edge tuples are the same tuples retrieved in step 2 or not. If these tuples are the same and the integrity check passed, then all values satisfying the query have been returned by the server.

#### IV. PERFORMANCE ANALYSIS

To assess the efficiency of our techniques we implemented a tool to evaluate the performance of using a Keyed-Hash

Message Authentication Code (HMAC), as the MAC function, and CMAC. The prototype was implemented using the C programming language and the OpenSSL library. The DBMS used was MySQL database and the experiments were performed in a machine running both MySQL server and client application. The machine had Intel Core 2 Quad CPU Q8400 with 4Mb cache, at 2.66GHz, 4GB RAM 800Mz, and 320Gb disk, SATAII, 16Mb cache, 7200RPM, running an Ubuntu 11.04 32-bit operating system with OpenSSL 0.9.8d and MySQL 5.1. Additionally, we used the SHA-1 hash function to calculate the HMAC with a 256-bit long key and disabled the cache of the MySQL.

We considered different scenarios to evaluate the performance of the proposed techniques. For each scenario, we executed the workload a thousand times over a table with 10 thousand tuples of random values. All the results shown below are the average of these executions. In all scenarios, we focus on evaluating the amount of time spent on the operations of INSERT, UPDATE, DELETE and SELECT, performed under four distinct conditions:

- 1) Without security mechanisms;
- 2) Using HMAC only;
- 3) Using both HMAC and CMAC;
- 4) Using both HMAC and CMAC in the circular mode.

#### Insert

In the first scenario, we focused on measuring and comparing the execution times for the INSERT operation under each specified condition. The results (as we can see in Figure 6) show that the baseline took  $42,3 \mu s$ , while the HMAC took  $47 \mu s$ , 90% of which is spent on the server side and 10% on the client side. The scenario with the use of CMAC executed in  $118,3 \mu s$ , with 91% of the time spent on the server and 9% on the client. The CMAC in the circular mode executed in  $331,7 \mu s$ , where 72% is executed by the server and 28% by the client.

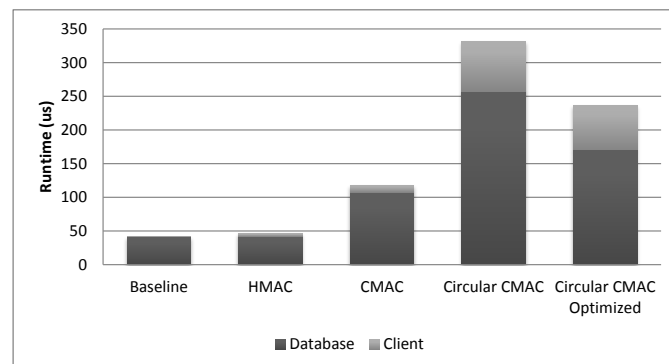


Fig. 6. Comparison of execution time of the INSERT operation

The CMAC in the circular mode can be optimized if the client store a small amount of data. The major reason for the difference between the the regular mode and the circular mode of the CMAC is that in the circular mode we need to retrieve and update additional rows. If the client stores the first row

locally, we eliminate one query, reducing the execution time from 331,7  $\mu$ s to 236,5  $\mu$ s.

*Update*

In the second scenario, we focused on measuring and comparing the execution times for the UPDATE operation under each specified condition. The results (shown in Figure 7) show that the baseline took 127,6  $\mu$ s, while the HMAC took 134  $\mu$ s, 95% of which is spent on the server side and 5% on the client side. The CMAC (both in regular and circular mode) executed in 381,9  $\mu$ s, with 80% of the time spent on the server and 20% on the client. The reason that the execution time for the CMAC in the regular and circular mode are the same is because they execute the exact same operations.

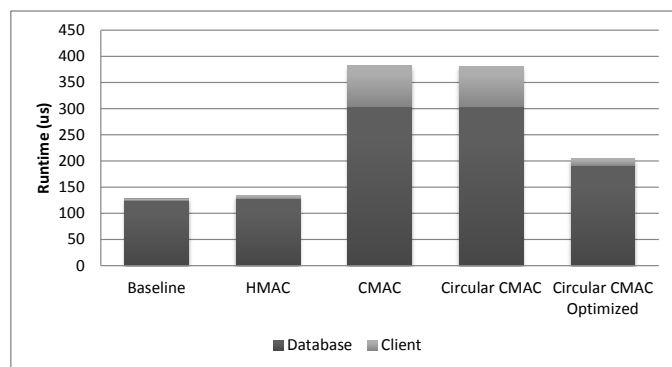


Fig. 7. Comparison of execution time of the UPDATE operation

We can also optimize the CMAC for the UPDATE operation if we consider that some values are available on the client side in the moment of the operation. In this case, when updating a row  $n$ , we need the MAC and CMAC of the  $n + 1$ -th and the  $n - 1$ -th rows (for example, these values could have been retrieved when the client retrieved the  $n$ -th row). If these rows are available on the client side in the moment of the update, the execution time is 204,5  $\mu$ s.

*Delete*

In the third scenario, we focused on measuring and comparing the execution times for the DELETE operation under each specified condition. The baseline executed in 51  $\mu$ s and when using the HMAC to delete a row, there is no additional cost since there is no extra operations to be performed (as shown in Tables I and II). On the other hand, the CMAC (both in regular and circular mode) executed in 186,5  $\mu$ s, with 96% of the time spent on the server and 4% on the client, as we can see in Figure 8. As we have shown for the UPDATE operation, the CMAC in the regular and circular mode have the exact same operations and therefore the overhead is the same.

We can use the same idea presented for the UPDATE operation to improve the efficiency of the CMAC. In the naive implementation, before deleting a row  $n$ , we execute a select query to retrieve the  $n + 1$ -th and the  $n - 1$ -th rows. Considering that these rows are available on the client side in the moment of the delete, the execution time is reduced from 186,5  $\mu$ s to 105,2  $\mu$ s.

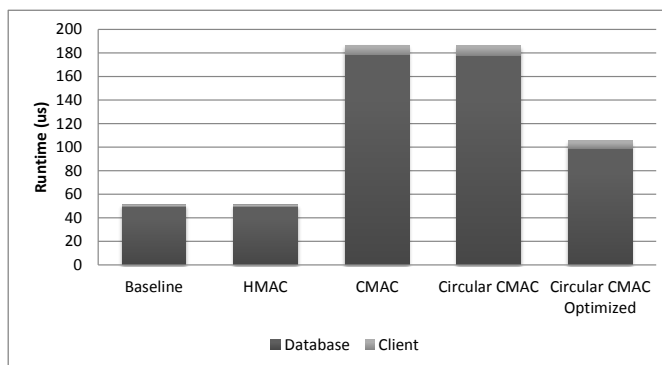


Fig. 8. Comparison of execution time of the DELETE operation

*Select*

Finally, in the last scenario, we focused on measuring and comparing the execution times to check the integrity during the SELECT operation under each specified condition. A SELECT query, without verifying the integrity (the baseline) took 18,4  $\mu$ s. To verify the integrity of the HMAC the client needs to recalculate the HMAC and compare it to the one retrieved from the server. This operation executed in 22,5  $\mu$ s, due to the calculation of the HMAC. When using the CMAC, the client needs to retrieve the HMAC of the previous row and recalculate both the HMAC and CMAC. These extra operations increase the execution time to 54  $\mu$ s, as we can see in Figure 9. However, if we consider that the previous row is available on the client side, the execution time is reduced to 27,6  $\mu$ s (for example, the client retrieved the  $n$ -th and  $n - 1$ th rows in a single query).

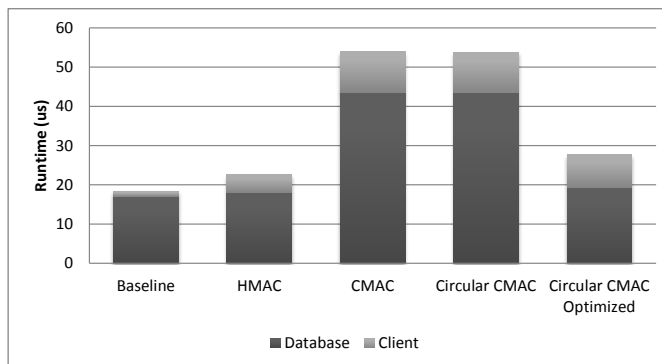


Fig. 9. Comparison of execution time of the SELECT operation

*Summarizing*

As we could see in the results, the impact of using each method is affected by two main factors: i) the number of sql operations; ii) the number of cryptographic functions performed. The number of sql operations performed by every type of action (ex: an update when using CHMAC requires two SQL operations) clearly has a bigger impact on the performance. The cryptographic-only operation have shown very low impacts. The table I shows the number of SQL

operations each method requires. The table II shows the number of cryptographic functions each method requires.

Although the cost of the SQL operations are 2-7 times greater with our method, it is more efficient than previous work. E. Mykletun et al. [13] and Narasimha and Tsudik [14] uses asymmetric cryptography, which is 1,000-10,000 times slower than simple MAC functions. In terms of the storage, our method generates less data to be stored in the database. For example, considering an RSA 2048-bit long key and the SHA-1 function, which are the values recommended by NIST [15] for 2014, our method generates approximately 13 times less data to control the integrity of the database.

Additionally, when comparing to the approaches based on authenticated structures [7], [8], [9], [4], our method has a lower complexity. That is, the cost of our method is constant to the database size ( $O(1)$ ) while the approaches based on authenticated structures are usually logarithmic ( $O(\log n)$ ). Therefore, for larger databases our method is more efficient.

It is important to notice that the DBMS cache was disabled while running the tests. In a real environment, with the cache enabled, the retrieval of previous row necessary to calculate the CMAC will be cached with a high probability, reducing the total cost of each SQL operation.

## V. FINAL REMARKS

This paper proposes secure and efficient methods for providing integrity and authenticity for relational database systems. Our methods focus on strategies for detecting unauthorised actions (insertions, deletions and updates) from a vulnerable database server.

Prior work either requires modifications in the database implementation or uses inefficient cryptographic techniques (for example, public key cryptographic). The requirement of modifying the core of a database system makes the deployment of these methods difficult in real world scenarios. Thus, one significant advantage of our method is that it is DBMS-independent and can be easily deployed in existing environments. Another advantage of our method is that we focused on using a more simple and efficient cryptographic algorithm to provide the integrity checks.

The performance requirements for each of our methods were presented and alternatives to minimise their costs and its consequences were discussed. Finally, we believe that the transparency and independency of our method makes it easily deployable and compatible with real world demands.

As a future work, we would like to address the roll-back attack. A roll-back attack is characterized when the attacker restores the database to a previous valid state. With this attack, the attacker can delete rows without being noticed, for example. Prior work, as well as this paper, are vulnerable to such attacks. Another interesting matter of research is to address the actions to be taken in case of a detected attack. We consider that the ideal solution is to restore the database to a valid state before the attack.

## REFERENCES

- [1] P. Samarati and S. D. C. di Vimercati, "Data protection in outsourcing scenarios: Issues and directions," in Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ser. ASIACCS '10. New York, NY, USA: ACM, 2010, pp. 1–14. [Online]. Available: <http://doi.acm.org/10.1145/1755688.1755690>
- [2] T. K. Dang, "Ensuring correctness, completeness, and freshness for outsourced tree-indexed data," *Inf. Resour. Manage. J.*, vol. 21, no. 1, Jan. 2008, pp. 59–76. [Online]. Available: <http://dx.doi.org/10.4018/irmj.2008010104>
- [3] I. Kamel, "A schema for protecting the integrity of databases," *Computers & Security*, vol. 28, no. 7, 2009, pp. 698–709.
- [4] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin, "Dynamic authenticated index structures for outsourced databases," in Proceedings of the 2006 ACM SIGMOD international conference on Management of data, ser. SIGMOD '06. New York, NY, USA: ACM, 2006, pp. 121–132. [Online]. Available: <http://doi.acm.org/10.1145/1142473.1142488>
- [5] T. Aditya, P. Baruah, and R. Mulkamala, "Employing bloom filters for enforcing integrity of outsourced databases in cloud environments," in Advances in Computing and Communications, ser. Communications in Computer and Information Science, A. Abraham, J. Lloret Mauri, J. Buford, J. Suzuki, and S. Thampi, Eds. Springer Berlin Heidelberg, 2011, vol. 190, pp. 446–460. [Online]. Available: [http://dx.doi.org/10.1007/978-3-642-22709-7\\_44](http://dx.doi.org/10.1007/978-3-642-22709-7_44)
- [6] T. Aditya, P. K. Baruah, and R. Mulkamala, "Space-efficient bloom filters for enforcing integrity of outsourced data in cloud environments," in Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing, ser. CLOUD '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 292–299. [Online]. Available: <http://dx.doi.org/10.1109/CLOUD.2011.40>
- [7] G. Di Battista and B. Palazzi, "Authenticated relational tables and authenticated skip lists," in Proceedings of the 21st annual IFIP WG 11.3 working conference on Data and applications security. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 31–46. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1770560.1770564>
- [8] A. Heitzmann, B. Palazzi, C. Papamanthou, and R. Tamassia, "Efficient integrity checking of untrusted network storage," in Proceedings of the 4th ACM international workshop on Storage security and survivability, ser. StorageSS '08. New York, NY, USA: ACM, 2008, pp. 43–54. [Online]. Available: <http://doi.acm.org/10.1145/1456469.1456479>
- [9] G. Miklau and D. Suciu, "Implementing a tamper-evident database system," in Proceedings of the 10th Asian Computing Science conference on Advances in computer science: data management on the web, ser. ASIAN'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 28–48. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2074944.2074951>
- [10] R. C. Merkle, "A certified digital signature," in CRYPTO, ser. Lecture Notes in Computer Science, G. Brassard, Ed., vol. 435. Springer, 1989, pp. 218–238.
- [11] W. Pugh, "Skip lists: a probabilistic alternative to balanced trees," *Commun. ACM*, vol. 33, no. 6, Jun. 1990, pp. 668–676. [Online]. Available: <http://doi.acm.org/10.1145/78973.78977>
- [12] R. d. S. M. Anderson Luiz Silvério and R. F. Custódio, "Efficient integrity checking for untrusted database systems," in Proceedings of the 28th Brazilian Symposium on Databases, ser. WTDBD'13. Sociedade Brasileira de Computação, 2013, pp. 36–42.
- [13] E. Mykletun, M. Narasimha, and G. Tsudik, "Authentication and integrity in outsourced databases," *ACM Transactions on Storage*, vol. 2, no. 2, May 2006, pp. 107–138. [Online]. Available: <http://portal.acm.org/citation.cfm?doid=1149976.1149977>
- [14] M. Narasimha and G. Tsudik, "Dsac: integrity for outsourced databases with signature aggregation and chaining," in Proceedings of the 14th ACM international conference on Information and knowledge management, ser. CIKM '05. New York, NY, USA: ACM, 2005, pp. 235–236. [Online]. Available: <http://doi.acm.org/10.1145/1099554.1099604>
- [15] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, "Recommendation for key management - part 1: General (revision 3)," National Institute of Standards and Technology, NIST Special Publication 800-57, Jul 2012. [Online]. Available: [http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57\\_part1\\_rev3\\_general.pdf](http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf)

TABLE I  
NUMBER OF SQL OPERATIONS PERFORMED IN EACH METHOD

	No protection	MAC	CMAC	Circular CMAC	CMAC Optimized
Insert	1	1	2	3	2
Update	2	2	4	5	2
Delete	1	1	3	3	2
Select	1	1	2	2	1

TABLE II  
NUMBER OF CRYPTOGRAPHIC OPERATIONS PERFORMED IN EACH METHOD

	No protection	MAC	CMAC	Circular CMAC	CMAC Optimized
Insert	0	1	2	3	3
Update	0	1	3	3	3
Delete	0	1	1	1	1
Select	0	1	2	2	2

- [16] M. Xie, H. Wang, and J. Yin, "Integrity auditing of outsourced data," *Very large data bases*, 2007, pp. 782–793. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1325940>
- [17] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '96. London, UK, UK: Springer-Verlag, 1996, pp. 1–15. [Online]. Available: <http://dl.acm.org/citation.cfm?id=646761.706031>
- [18] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," RFC 2104 (Informational), Internet Engineering Task Force, Feb. 1997, updated by RFC 6151. [Online]. Available: <http://www.ietf.org/rfc/rfc2104.txt>

# Quantifying the Elasticity of a Database Management System

Christian Tinnefeld, Daniel Taschik, Hasso Plattner

Hasso Plattner Institute

University of Potsdam, Germany

{firstname.lastname}@hpi.uni-potsdam.de

**Abstract**—There exist different and well-established approaches for quantifying the performance of a database management system. With the advent of provisioning information technology infrastructure over the Internet, the aspect of elasticity became more important as it defines how well a system adapts to a changing workload. For a database management system there is no commonly agreed approach or model how to quantify its elasticity. In contrast, the cloud storage system (NoSQL) community developed several approaches how to measure elasticity. In this paper we contribute by I) presenting an extensive review of the existing approaches for measuring the elasticity of NoSQL systems, II) compare their parameters and used metrics, III) transfer the lessons learned and introduce a model for quantifying the elasticity of a database management system.

**Keywords**-Elasticity, Database Management System, NoSQL

## I. INTRODUCTION

Minhas et al. say that elasticity is the “ability to grow and shrink processing capacity on demand, with varying load” [1]. Another definition is given by Agrawal et al. which state that elasticity is “the ability to deal with load variations by adding more resources during high load or consolidating the tenants to fewer nodes when the load decreases, all in a live system without service disruption” [2]. A more general definition has been given by the National Institute of Standards and Technology: “Capabilities can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out, and rapidly released to quickly scale in. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time” [3]. Figure 1 illustrates frequently used terminology in conjunction with elasticity. After this introduction follows Section 2 which reviews state-of-the-art elasticity benchmark for NoSQL systems and compares the used

metrics, how a scale-out is triggered, how data migration is done, how the operational costs are quantified, which query and workload characteristics are applied and how the scale-out is managed. Section 3 then continues with a model that allows to quantify the elasticity of a relational database management system. The model is based on measuring the query processing latency in combination with a breakdown of the utilized hardware resources. Section 4 close with the conclusions and presents the future work.

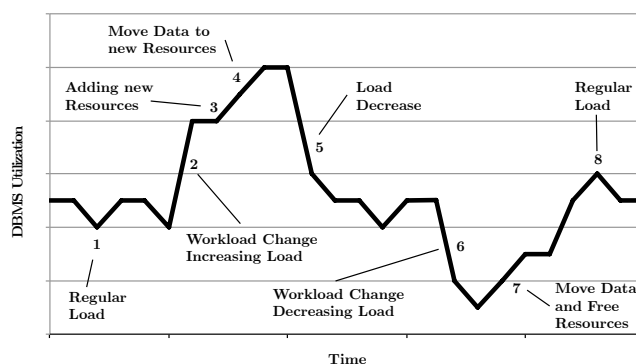


Figure 1: Terminology used in conjunction with elasticity

## II. ELASTICITY BENCHMARKS FOR NoSQL SYSTEMS

A catalogue of metrics for evaluating cloud services is presented in the work of Li et al. [4]. The metrics presented for elasticity are split up in three groups. (1) Resource acquisition time, (2) resource release time and (3) cost and time effectiveness. The first group describes the time a resource takes to be available for the system from the moment it has been requested until the moment of availability to the system. The second group describes the time to release an unnecessary cloud resource. It can be split into the time to remove the existing deployment

and the time to stop the cloud resource and finally release it. The third group of metrics describes the relation between the costs and the runtime of provisioned cloud resources. Especially the third group has been recognized in our framework considerations by measuring the runtime costs from the moment of provisioning a resource to the moment of de-provisioning.

Weinman [5] suggests a model for calculating elasticity. He states that almost every business tries to match demand and supply. By defining a function  $D(t)$  representing a mapping from demand to a resource in time and a second function  $R(t)$  representing allocated resources over time, he proposes that a perfect capacity strategy is given, when  $D(t) = R(t)$ . In this case, the resources for matching an existing demand are available in the right amount. Too little resources do not create loss in revenue neither excess resources create needless costs. He introduces a function describing the financial loss an unmet demand can cause. That costs unnecessarily accrue in situation of excess resources. He also evaluates how monitoring interval and provisioning time for a resource influences the negative impact.

Weinman's paper had a great influence on the work of Islam et al. [6]. After the definition of elasticity in the context of cloud platforms, the authors propose a concept on how a consumer can quantify the elasticity of these platforms. The concept is build on the idea of Weinman and extends its calculation model with a penalty for over- and under-provisioning. Hereby, a differentiation between allocated and charged resources takes place. By normalizing the calculated value, the authors propose a single metric of elasticity for a cloud-based platform. They showcase their approach for an elasticity measurement environment and make use of it for different consumer specific workloads scenarios. The work presented in the above mentioned paper, served as inspiration for the elasticity benchmark framework for relational database management systems in this thesis. The provisioning based calculation model is taken from Islam et al. and has been slightly modified to fit the needs for a relational database management system (DBMS).

One of the most known and famous NoSQL benchmark is the *Yahoo! Cloud Serving Benchmark* (YCSB) [7]. It has been developed at Yahoo to help developers to choose which cloud based data storage might be the best for their workload. It provides a two-tier structure. The first one concentrates on performance whereas the second and more interesting one looks at scalability. Elastic

speed-up is measured by monitoring the performance of a system as the number of machines is increased while running a constant workload. A good elastic system must show an improvement in performance. A short disruption in service is accepted while the system is reconfiguring itself. Elasticity itself is not quantified and only the impact of read latency is considered.

Konstantinou et al. [8] conducted a study on the costs and efficiency of adaptive expansion and contraction of NoSQL databases over a cloud platform. The authors took three popular NoSQL representatives (HBase, Cassandra and Riak) and performed experiments with four YCSB workloads. They analyzed how the cloud data storages performed by measuring query throughput, mean query latency as well as CPU and memory consumption during a stress test. Costs for the initialization and reconfiguration of nodes as well as rebalancing of data within the cluster are ascertained in terms of time and data volume. Finally, Konstantinou et al. present a framework for monitoring and automating cluster resize operations. The authors benchmarked only NoSQL systems and did not consider analytical workloads. They did not track financial aspects for operating the platform. Thibault Dory et al. [9] introduce a dimensionless measure for elasticity for cloud databases. In their methodology, Dory defines elasticity as a characterization of how a cluster reacts on node provisioning. Regarding to Dory, elasticity is defined by two properties. The first one is the time a cluster takes to stabilize itself after nodes have been added. The second property is the influence on the cluster's performance. In regard to the first property, they define a cluster as stable when the variations of time needed to fulfill a certain number of requests is equivalent to the variations of a cluster known as being stable. In this case, it is a cluster where no data is being moved from one node to another. The authors suppose that the response time for requests increases after new nodes have been added, and then decrease after a certain amount of time. Dory et al. define the elasticity as a ratio of elastic overhead to the absolute performance of a cluster. This approach of quantifying elasticity is validated against a NoSQL architecture with an OLTP workload using the behavior of a Wikipedia user.

Another cloud-based quality measurement and analysis framework is introduced in the work of Klems et al. [10]. By contributing with an infrastructure, configuration and cluster configuration manager, Klems et al. propose a framework to evaluate the performance, latency and

consistency of the cloud-based data stores Amazon S3, Amazon SimpleDB, DynamoDB and Cassandra using the Yahoo! Cloud Serving Benchmark and YCSB++ benchmark. The authors analyze different scaling strategies and conflicts between contradictory objectives, such as consistency versus high-availability and scalability. In addition, they examine the impact of system changes on performance and availability. Unfortunately, further investigations of elastic scalability in regard to data migration and performance impacts of such, is not presented, though can be expected in future work.

The Cloud Service Measurement Index Consortium proposed a framework [11] to define and measure certain quality of service aspects of cloud providers. Its service measurement index is intended to help customers to rank and compare cloud service providers based on customers requirements like accountability, agility, costs, performance, assurance, security and usability. Elasticity, defined as how much a cloud service is able to scale during peak time, is seen as a subcomponent of agility. As part of a case study, Garg et al. present a relative service-ranking vector for elasticity. This approach takes the time a system needs to expand or contract into account. An impact on how data migration might influence the performance is not considered. The remainder of this section gives a detailed breakdown of the different parameters. Table I on the next page summarizes which parameters are considered in the previously mentioned related work.

#### A. Metrics

Each of the three benchmarks uses different metrics to express elasticity. Islam's [6] approach is based on a financial penalty model. When a system runs in an undesired state, being either under- or over-provisioned a fine will be charged. Undesired in this context means that either there are not enough resources to handle the load or that there are too many, unnecessary resources present which could be de-provisioned for the cause of cost savings. The penalty amount is calculated from the time the system is in an undesired state, representing the responsiveness of the system to scale and change the state into a desired one. The smaller the penalty the more elastic is the system.

The Yahoo! Cloud Serving Benchmark [7] uses quite a basic metric for elasticity. It takes response latency to requests as a measure to express elastic speedup. The authors conducted a benchmark examining the elastic

speedup of three cloud-based data stores. For each data store they started with a small cluster offering a load feasible for a three-times bigger system. Then, they added nodes to the cluster until it was stabilized and able to serve the load. The implications on latency have been recorded and taken as a degree of elastic speedup. Dory et al. [9] use a dimensionless metric for elasticity whereas Konstantinous et al. [8] use throughput, latency as well as CPU utilization as their elasticity measure. In our opinion, taking just the latency is not sufficient enough to measure elasticity. I argue that there are more metrics influencing the elasticity of a DBMS than just the responsiveness to queries. Taking fixed budgets for a cloud system or energy consumption of the hardware into account, it is conceivable that there are more dimensions expressing elasticity. I am in favor of Islam's approach, taking the costs per time as a measurement. Because pay-as-you-go is one of the advantages of cloud computing over established server based systems, taking the runtime costs of a system into account creates a more meaningful metric for elasticity.

#### B. Scaling Trigger

The various benchmarks and frameworks trigger a scaling operation differently. Whereas YCSB and Dory et al. use static scaling, meaning it is triggered either manually or at a fixed moment during the benchmark, Konstantinous and Islam take CPU utilization of the system as scaling trigger. Konstantinous et al. perform a scale-out as soon as one node has more than 40% CPU utilization. A scale-in, on the other hand, is done as soon as the average of all nodes are less than 15% utilized. Islam triggers scaling operations in the event of an undesired provisioning state (under- or over-provisioning).

I emphasize the fact that a framework should be able to trigger scaling operations automatically. In real-world scenarios, DBMS operators use automated tools for scaling instead of issuing manual scaling instructions.

#### C. Data Migration

Migrating data from an existing cluster to an added node has an impact on elasticity, because transferring data between nodes or throughout an entire cluster takes time and slows down regular operations. Except for Islam et al., all other frameworks consider the time needed for the migration operation. Konstantinous et al. also observe the amount of the moved data. This enables a better differentiation between different architectural



TABLE I: Overview of Elasticity Benchmarks and Frameworks for NoSQL Systems

	Properties	Islam et al.	YCSB	Dory et al.	Konstantinous et al.
Metrics	Throughput	-	-	-	x
	Latency	-	x	-	x
	CPU Usage	-	-	-	x
	Monetary	x	-	-	-
	Dimensionless	-	-	x	-
Scaling Trigger	Static	-	x	x	-
	Latency-based	x	-	-	-
	CPU Utilization	x	-	-	x
Consideration of Data Migration	Time	-	x	x	x
	Amount	-	-	-	x
Operating Costs	Taken into Account	x	-	x	-
Query Characteristic	OLTP	x	x	x	x
	OLAP	-	-	-	-
Workload Characteristics	Sinus Shaped	x	x	-	-
	Plateau Shaped	x	x	-	-
	Exponential Shaped	x	x	-	-
	Linear	x	x	x	x
	Random	x	x	-	-
	Zipfian	-	x	-	x
Scale Management (Monitoring, Cloud/Cluster Management, Rebalancing)	Provide Toolset	-	-	-	x
	External Tools (e.g. Amazon Autoscale)	x	-	-	-
	Manual	-	x	x	-
Applicability Difficulty	Simple	x	x	-	x
	Difficult	-	-	x	-

characteristics of databases. I comply that the time for adding a new node and getting it ready to serve should be reflected in an elasticity benchmark. In addition, it is of interest to track the time for data migration and the time for provisioning computing resources, booting a node, starting the DBMS instance and registering it with the existing cluster.

#### D. Operational Costs

Elasticity is a time-critical property. The faster a system is able to adapt, the more it is considered elastic and this has consequences on operational costs for the system, e.g. if computing resources are provisioned and not used, the landscape generates more costs than necessary to serve the load. Islam et al. and Dory et al. pay attention to the operational costs in their elasticity determination. As already mentioned in the metrics part of this section, pay-as-you-go is the reason why operational costs have to be regarded and must be considered in an elasticity benchmark.

#### E. Query & Workload Characteristics

All four presented frameworks execute only transactional queries in their benchmarks. The applied workload patterns are different. Islam et al. as well as YCSB have a huge variety of different workload patterns available. Dory et al. and Konstantinous et al. use a linear workload and Konstantinous makes use of an additional zipfian workload. Nevertheless, all frameworks lack a real-world workload scenario. YCSB allows customizing and implementing industry-related scenarios. The benchmark must reflect a workload pattern, which simulates the intended area of deployment of the DBMS. This can be a simple sinus-like workload, but in an enterprise environment it is conceivable to run complex analytical workloads.

#### F. Scale Management

To measure elasticity properly, it is necessary for the framework to provide tools for monitoring load, to add and remove nodes, to redistribution of data and to manage the cluster. The work of Konstantinous et al. is the only framework that provides a conventional

toolset. Islam et al. are using tools from a third-party provider, in this case, Amazon CloudWatch and Auto Scaling. It can only be assumed that monitoring and provisioning is handled manually in the work of Dory et al. and YCSB, because scaling is not automated at all. In our opinion, a proper framework must provide these tools to supplement missing features in the DBMS. Only without manual intervention, realistic measurements can be conducted.

### III. QUANTIFYING THE ELASTICITY OF A DATABASE MANAGEMENT SYSTEM

After extensively reviewing approaches for quantifying the elasticity of a NoSQL system, we present an elasticity calculation model for relational DBMS which is inspired by the work of Islam et al. [6]. To determine elasticity, Islam et al. offer a way for consumers to measure elasticity for cloud platforms by defining financial penalties for over- and under-provisioning. Under-provisioning means that the system has less computing resources available than actually necessary to fulfill all requests (demand) against the system in a desired time. Over-provisioning, on the other hand, describes the state of a system in which there are excess resources available than really needed to fulfill the demand against the system. These excess resources lead to higher runtime costs that are avoidable. The financial reflection model of Islam et al. is taken and got adapted to fit the needs for an elasticity benchmark for relational DBMS. To build a calculation model for elasticity, a few assumptions need to be stated. Elasticity enables very cost-efficient operation. Therefore, costs play a key role in defining a metric for elasticity. The costs for a utilized node to run for one hour are defined as 100 cents. The price is derived from the Amazon AWS EC2 pricing list [12] for an *Amazon M3 Double Extra Large Instance*. The hardware sizing of this instance type is powerful enough to run a relational DBMS and therefore the price can be valued as reasonable. At Amazons AWS EC2, a resource is allocated for a minimum time period of one hour. To avoid complexity in situations where a resource is allocated and therefore charged but not available, because it is already de-provisioned or not yet booted, the chargeable time period is reduced to one second. Consequently, the calculation of chargeable supply as used by Islam et al. is discarded. The moment the resource is requested, it is charged until the second it is de-provisioned.

To calculate the elasticity for an elastic RDBMS, it is nec-

essary to sum up the number of used nodes. A resource can be charged from the moment of provisioning of a node to the moment a service on that node is serving, because booting time of a prepared image with all required services pre-installed can be done in a constant time. Hence, for easier consideration it is assumed that the chargeable time begins at the moment the framework detects an under-provisioned state and ends at the moment the framework de-provisions the node. The already mentioned penalties for over- and under-provisioning need to be specified as well. An under-provisioning penalty of 10 cents for every second in an under-provisioned state is specified. This reflects six times the costs of an additional node. As Weinman et al. [5] emphasize, the benefit of using resources should clearly outweigh the costs of it. The state of over-provisioning is not penalized because too many provisioned resources create avoidable costs, which are thereby treated as a penalty. So only the pure costs of the provisioned resources are taken into account. In summary, the following numbers need to be ascertained to calculate elasticity: maximum allowed latency, number of used DBMS nodes, runtime of utilized RDBMS nodes, time while being under-provisioned.

The maximum allowed latency is configured by the benchmark executor and describes in this case the longest acceptable response time for a benchmark suite run. A benchmark suite run is defined as a set of queries, which get executed by a benchmark client. The benchmark client repeats a benchmark suite run over and over until the framework controller is stopping the client. The amount and runtime of server nodes can be retrieved from the cluster management controller that is responsible for provisioning of RDBMS server nodes. The time while the RDBMS is under-provisioned can be gathered by taking the latency until a benchmarking suite run has finished and subtract the maximum allowed latency for a benchmark suite run as demonstrated in Formula 1.

$$f(t) = \text{Benchmark\_runtime}_t - \text{upper\_threshold}_t \quad (1)$$

To get the time when the benchmark suite runtime was above the upper limit, the function  $f_{\text{cutoff}}(t)$  needs to be ascertained. Therefore, only values above the limit as represented in Formula 2 are taken into account.

$$f_{\text{cutoff}}(t) = \begin{cases} f(t) & \text{if } f(t) > 0 \\ 0 & \end{cases} \quad (2)$$

Now, that only the amount of time, when the runtime of the benchmark suite has taken longer than the upper bound per node has been determined, the accumulation over all recorded instances can be done as seen in Formula 3

$$P(t_0, t_{\text{end}}) = \sum_0^{\#instances} \int_{t_0}^{t_{\text{end}}} f_{\text{cutoff}}(t) dt \quad (3)$$

The next step is to ascertain the penalty sum for all instances. The sum will be multiplied with the defined penalty amount. This results in an overall penalty as seen in Formula 4.

$$P = P(t_0, t_{\text{end}}) \times p_{\text{under-provisioned}} \quad (4)$$

Finally, the penalty and the runtime costs will be aggregated. To calculate the runtime costs, the number of running nodes for a certain time frame needs to be multiplied by the costs of it. Formula 5 shows the calculation of it.

$$C_{\text{nodes}} = \left( \sum_{n=1}^{\#nodes} runtime(n) \right) \times c_{\text{node}} \quad (5)$$

Formula 6 shows the calculation of the final elasticity by adding up the penalty and the runtime costs for the provisioned resources  $C_{\text{nodes}}$  and dividing it by the runtime of the experiment. This results in a comparable value in cents per seconds.

$$E = \frac{P + \#nodes_{\text{charged}} \times c_{\text{nodes}}}{t_{\text{end}} - t_0} \quad (6)$$

The apparent significance of time-to-serve for an elastic relational DBMS does not need to be measured explicitly. It is implicitly provided by the runtime of the benchmarking suite. The longer it takes for a node to be ready to serve, the longer the cluster stays in an under-provisioned state. This results in a much higher penalty than for systems with a very low time-to-serve value.

#### IV. CONCLUSIONS AND FUTURE WORK

In this paper, known quantification models and methods from NoSQL systems are evaluated and taken into account to propose a model for quantifying the elasticity of a database management system. Features and characteristics as well as pre-conditions for elasticity

measurements are identified, presented and defined. The proposed elasticity calculation model is a provisioning based quantification model. The corner stones of the model are the aspects of maximum allowed latency, number of used DBMS nodes, runtime of utilized RDBMS nodes and the time while the DBMS is being under-provisioned.

The presented elasticity model enables relevant elasticity determination experiments yielding unique comparable values for a relational DBMS under a specific workload. This has to be demonstrated in future work by conducting a case study. Here, different workloads will be executed on different relational DBMSs and the resulting elasticity will be calculated and compared.

#### REFERENCES

- [1] U. F. Minhas, R. Liu, A. Abounaga, K. Salem, J. Ng, and S. Robertson, "Elastic Scale-Out for Partition-Based Database Systems," 2012 IEEE 28th International Conference on Data Engineering Workshops, 2012, pp. 281–288. [Online]. Available: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6313694>
- [2] D. Agrawal, A. Abbadi, S. Das, and A. Elmore, "Database scalability, elasticity, and autonomy in the cloud," in Database Systems for Advanced Applications, ser. Lecture Notes in Computer Science, J. Yu, M. Kim, and R. Unland, Eds., vol. 6587. Springer Berlin Heidelberg, 2011, pp. 2–15. [Online]. Available: [http://dx.doi.org/10.1007/978-3-642-20149-3\\_2](http://dx.doi.org/10.1007/978-3-642-20149-3_2)
- [3] P. Mell and T. Grance, "The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology," National Institute of Standards and Technology NIST, Gaithersburg, MD, vol. 145, 2011.
- [4] Z. Li, L. O'Brien, H. Zhang, and R. Cai, "On a Catalogue of Metrics for Evaluating Commercial Cloud Services," 2012 ACM/IEEE 13th International Conference on Grid Computing, 2012, pp. 164–173. [Online]. Available: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6319167>
- [5] J. Winman, "Time is Money : The Value of On-Demand," <http://www.joeweinman.com>, pp. 1–29, 2011.
- [6] S. Islam, K. Lee, A. Fekete, and A. Liu, "How a consumer can measure elasticity for cloud platforms," Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering - ICPE '12, 2012, p. 85. [Online]. Available: <http://dl.acm.org/citation.cfm?doid=2188286.2188301>
- [7] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10, 2010, p. 143. [Online]. Available: <http://portal.acm.org/citation.cfm?doid=1807128.1807152>

- [8] I. Konstantinou, E. Angelou, C. Boumpouka, D. Tsoumakos, and N. Koziris, "On the elasticity of nosql databases over cloud management platforms," in Proceedings of the 20th ACM International Conference on Information and Knowledge Management, ser. CIKM '11. New York, NY, USA: ACM, 2011, pp. 2385–2388. [Online]. Available: <http://doi.acm.org/10.1145/2063576.2063973>
- [9] T. Dory, B. Mejias, P. V. Roy, and N. L. Tran, "Measuring elasticity for cloud databases," in CLOUD COMPUTING 2011: Proceedings of the The Second International Conference on Cloud Computing, GRIDs, and Virtualization, I. 978-1-61208-153-3, Ed., 2011, pp. 154–160.
- [10] M. Klems, D. Bermbach, and R. Weinert, "A Runtime Quality Measurement Framework for Cloud Database Service Systems," 2012 Eighth International Conference on the Quality of Information and Communications Technology, 2012, pp. 38–46. [Online]. Available: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6511780>
- [11] S. K. Garg, S. Versteeg, and R. Buyya, "A framework for ranking of cloud computing services," Future Generation Computer Systems, vol. 29, no. 4, 2013, pp. 1012–1023. [Online]. Available: <http://linkinghub.elsevier.com/retrieve/pii/S0167739X12001422>
- [12] Amazon, "Amazon EC2 Pricing List," 2013. [Online]. Available: <http://aws.amazon.com/de/ec2/pricing/>

# Hierarchical Piecewise Linear Approximation

## A Novel Representation of Time Series Data

Vineetha Bettaiah, Heggere S Ranganath

Computer Science Department  
The University of Alabama in Huntsville  
Huntsville, USA  
vineetha.bettaiah@gmail.com

**Abstract**— This paper presents a Hierarchical Piecewise Linear Approximation (HPLA) for the representation of time series data in which the time series is treated as a curve in the time-amplitude image space. The curve is partitioned into segments by choosing perceptually important points as break points. Each segment between adjacent break points is recursively partitioned into two segments at the best point or midpoint until the error between the approximating line and the original curve becomes less than a pre-specified threshold. The HPLA achieves dimensionality reduction while preserving prominent local features and general shape of the time series. The HPLA permits coarse-fine processing, allows flexible definition of similarity between two time series based on mathematical measures or general time series shape, and supports query by content, clustering and classification based on whole or subsequence similarity.

**Keywords**-Data Mining; Dimensionality Reduction; Piecewise Linear Representation; Time Series Representation.

### I. INTRODUCTION

Many areas of science, engineering, and business are generating, archiving and processing vast amounts of data. Because of the sheer volume, the data science community is challenged to develop new methodologies for the modeling, representation, retrieval, processing, understanding, and visualization of “Big Data”. Big data is a collection of larger and complex data sets, difficult to manage and process using traditional data management and processing techniques. One type of data that has received a lot of attention in recent years in diverse areas including medicine, astronomy, geology, atmospheric and space science, engineering, and financial markets is time series data. Mathematically, a time series  $T = \{x_1, x_2, \dots, x_n\}$  is a sequence of  $n$  real numbers in the increasing order of time, where each value has a time stamp. The time spacing between adjacent samples  $x_i$  and  $x_{i+1}$  may remain constant or vary over the duration of the time series.

The main processing tasks or operations associated with time series data are query by content, clustering, classification, prediction, anomaly detection, motif discovery, and rule discovery [1]. These are well known problems in pattern recognition and data mining areas for many years. However, the proven pattern recognition and data mining methods are not suitable for processing time series data, mainly because of three reasons. First, the

dimensionality of time series is very high, could be as high as tens of thousands. Secondly, the corresponding elements of two time series may not align due to difference in length, scale, translation, shift or non-uniform spacing between adjacent elements. Finally, the notion of similarity in the context of time series is very different from the one used in pattern recognition. Unlike in pattern recognition, where all elements of pattern vectors are used to determine similarity between two patterns, only subsets of elements of the two time series may be used to determine their similarity. Two time series may be considered similar if they contain similar subsequences of sufficient length or several similar patterns in the same time order.

An obvious solution to the above problem is to use compact representations of time series that are capable of achieving a significant reduction in dimensionality without losing important features present in the original data. During the past two decades several piecewise linear, symbolic, transform and model based representations have been developed [2]-[5]. Each representation has its own advantages and disadvantages. For example, transform based representations being global representations do not provide local information about subsequences [6]. The symbolic representations, such as SAX lose most of the shape information due to two levels of approximation [4]. The Singular Value Decomposition (SVD) [3] and Hidden Markov Model (HMM) [5] representations are computationally very expensive. To the best of our knowledge, as of now, characteristics of an ideal (good) time series representation based on the needs of time series data mining applications are not explicitly identified.

This paper makes three contributions. First, in Section II, requirements an ideal time series representation should satisfy are identified. Secondly, in Section III, widely used piecewise linear representations are analyzed to determine their strengths and weaknesses by using the requirements identified in Section II as metrics. Thirdly, in Section IV, a new representation called Hierarchical Piecewise Linear Approximation (HPLA), which is closer to the ideal representation than existing representations, is described. The advantages of the HPLA representation are described in Section V. By using the compression ratio and representation accuracy as metrics, a comparison of the HPLA with Piecewise Aggregation Approximation (PAA) and Piecewise Linear Approximation (PLA) is given in Section VI.

Conclusions and recommendation for future research are given in Section VII.

## II. CHARACTERISTICS OF AN IDEAL TIME SERIES DATA REPRESENTATION

In this section, characteristics for an ideal time series data representation are identified based on the needs of the important time series data mining applications.

In *query by content*, the objective is to retrieve time series from the database that are similar in information content to the given query time series  $Q$  [1]. The similarity between  $Q$  and time series  $T$  in the database may be determined by matching  $Q$  and  $T$ , or sub-sequences of  $Q$  and  $T$ . Though the content is almost always specified by a query sequence, it is desirable to have flexibility on how the content is specified. For example, general shape of time series, sequence of events, and similar subsequences are valid specifications of content in many applications. *Therefore, to support query by content, the representation should support broad specification of content, and should have a distance measure satisfying lower bound criterion.*

In *clustering*, given a set of  $N$  unlabeled time series  $T_1, T_2, T_3, \dots, T_N$ , the goal is to partition the set into  $K$  groups based on a meaningful similarity measure, such that members belonging to a group are similar to one another, and members belonging to different groups differ significantly from one another [1]. The feature-based methods compute a small number of features to represent each time series, and then use clustering algorithms, such as  $k$ -means algorithms to cluster feature vectors. The model-based methods extract a set of parameters for each time series, and then find clusters by clustering parameters. The raw-data-based methods are rarely used due to high dimensionality. Today, clustering algorithms are set in a mathematical framework, which use feature vectors or model parameters. A syntactic clustering approach using broad similarity as perceived by humans is needed for clustering time series data. *Therefore, the representation should preserve salient attributes of the time series to support the development of mathematical and syntactic clustering algorithms.*

*Classification* is the process of assigning an input time series to one of the several known classes or categories [1]. Bayesian classifiers are not practical for use with raw time series as the computation of probability density functions for such high dimensionality time series is not feasible. The linear classifiers (perceptron, least mean square methods, support vector machines, etc.), and non-linear artificial neural networks require large number of samples, at least two times the dimensionality of time series. Even if the required training samples are available, training classifiers with large number of high dimensional vectors is not practical. Thus, classification based on features appears to be the only practical solution. *Therefore, the representation should preserve salient attributes of the time series, and allow the computation of geometric and mathematical features needed for training classifiers.*

Given a time series  $T = \{x_1, x_2, \dots, x_n\}$ , *prediction* is the task of determining likely values of  $x_i$  for  $i > n$ . The future

values are predicted based on the current evolution trend observed or mathematical models such as Hidden Markov Model developed from historic data similar to the current time series [1]. Usually, the model is based on prominent features of time series. *Therefore, the representation should preserve local features and evolution trends of time series as accurately as possible.*

*Motif detection* is the process of identifying an approximately repeating subsequence representing meaningful pattern in a time series or a group of time series [1]. Motifs have been widely used for rule-discovery, clustering and classification of time series data. *Therefore, the time series representation must facilitate the identification of motifs of varying lengths by preserving perceptually important points and local trends.*

*Rule discovery* learns temporal rules that are hidden or not obvious in time series data [1]. One approach is to transform time series to a sequence of symbols and use association rule mining algorithms to discover rules. Another approach is to transform time series to a sequence of events, and use classification trees to discover temporal rules. *Therefore, it should be possible to obtain from the representation a meaningful sequence of symbols and events as defined by the user.*

In addition to the application specific requirements identified above, a few general requirements are also listed below.

- 1) The representation should be as compact as possible to achieve maximum dimensionality reduction, and at the same time should allow the reconstruction of the original time series with little error.
- 2) The representation should allow matching two time series using full sequences or subsequences even if the two time series differ in length, scale, amplitude, and translation.
- 3) The representation should retain salient attributes and local evolution trends.
- 4) The representation should allow the computation of geometric and mathematical features, and model parameters to support feature and model based processing.
- 5) To support query by content, the representation should support broad and flexible specification of content.
- 6) The computation for building the representation itself should be reasonable, should not require prior knowledge of the type of motifs or general shape of the time series.
- 7) The representation should have a distance measure satisfying lower bound criterion.

## III. RELATED WORK

The time series representations can be broadly classified into four categories, namely, piecewise linear representations, transform based representations, symbolic representations, and model based representations. As the HPLA representation proposed in this paper is a piecewise linear approximation, only the piecewise linear representations are briefly described and analyzed.

### A. Piecewise Aggregation Approximation (PAA)

Let  $X = \{x_1, x_2, \dots, x_n\}$  be a time series of length  $n$ . The PAA representation of  $X$  is obtained by partitioning  $X$  into  $N$

segments of equal length  $n/N$ , where  $N \ll n$ , and then representing each segment by the mean of elements belonging to the segment [2].

**B. Piecewise Linear Approximation (PLA)**

The PLA is the most frequently used representation in which a time series  $X$  of length  $n$  is partitioned into  $N \ll n$  sections and each section is represented by a straight line. Keogh and Pazzani refer to the process of generating PLA representation as segmentation of the time series [7]. Uniform segmentation produces segments of equal length  $l = n/N$ . Non-uniform segmentation partitions the time series into segments of unequal length to best fit the shape of the time series. Linear interpolation approximates the segment  $X[a:b]$  by the line joining  $x_a$  and  $x_b$ . Linear regression fits the best possible line to  $X[a:b]$  in the least square sense. As linear interpolation requires constant time, it is the most widely used method.

Several variations of PLA representations have been developed in recent years. Yan et al. and Pratt et al. segment the time series at important maxima and minima, and represent the time series by a polyline joining adjacent local maximum and minimum [8][9]. Park et al. partition the time series into monotonically increasing or decreasing segments, and characterize each segment by a six dimensional feature vector [10]. Zhou et al. suggest building a PLA representation (Slope Threshold Change) by using points at which slope changes significantly as break points [11]. In Piecewise Linear Aggregate Approximation (PLAA), Nguyen Quoc et al. divide the time series into  $N$  segments of equal length, and represent each segment by the mean and slope of the best fitting straight line [3].

**C. Adaptive Piecewise Constant Approximation (APCA)**

The APCA representation of a time series is obtained by segmenting the time series into  $N$  segments of unequal length based on data. Long segments are used to represent data regions of low activity, and short segments are used to represent regions of high activity. Each segment is represented by its mean value and the index of the right end point. Therefore, the time series  $X = \{x_1, x_2, \dots, x_n\}$  is represented as  $\{\langle xv_1, xr_1 \rangle, \dots, \langle xv_N, xr_N \rangle\}$ , where  $xv_i$  is the mean of all values in the  $i^{th}$  segment, and  $xr_i$  is the index of the right most element of the  $i^{th}$  segment.

An evaluation of PLR representations based on requirements identified in Section II is given below.

1) The compression ratio, reconstruction accuracy and shape complexity of time series are related. The only way of achieving high reconstruction accuracy is by partitioning the time series into a large number of segments, which limits the extent to which the dimensionality is reduced. The APCA and the PLA representations achieve higher compression than PAA as they limit the number of segments by placing long segments in regions, where values are fairly constant or linear, respectively. For a given compression ratio PLA achieves higher reconstruction accuracy than PAA and APCA [14].

2) The orders of computation for PAA, PLA and APCA representations are  $O(n)$ ,  $O(nL)$ , and  $O(n \log_2 n)$ , respectively [2][7][14].

3) The PAA and APCA representations do not provide any information regarding the shape of the time series within segments. Therefore, there may not be sufficient information to detect shapes and trends spanning one or more segments, and the computation of many features needed for feature based data mining applications may not be possible. The PLA is better than PAA and APCA in approximating the shape of the time series, especially if perceptually important points are used as break points during segmentation.

4) The PAA and APCA representations do not use perceptually important points like local maxima and minima as break points. As a result, matching two time series which differ in length, scale, or translation is not easy or even possible. These representations are not suitable for establishing similarity between two time series based on their subsequences. It may be possible to deal with differences in length, scale, amplitude, translation, and subsequence matching using PLA representation if time series are segmented at perceptually important points.

5) For query by content application, the PAA and APCA have distance measures that satisfy minimum bounding criterion. However, specification of content based on shape and subsequences is not possible. In general, PLA does not have distance measure that satisfies minimum bounding criterion. Broad specification for content is possible only if time series are segmented appropriately.

The findings are summarized in Table I. “Yes”, “No”, and “May be” are used to indicate that the requirement is well satisfied, not satisfied, or partially satisfied, respectively. The PLA representation, if break points include

TABLE I. EVALUATION OF REPRESENTATION BASED ON REQUIREMENTS OF AN IDEAL REPRESENTATION

Requirement	PAA	PLA	APCA
1	No	Yes	No
2	No	Yes	No
3	No	Yes	No
4	No	Yes	No
5	No	Yes	No
6	$O(n)$	$O(n)$	$O(n \log_2 n)$
7	Yes	Yes	Yes

perceptually important maxima and minima, is expected to achieve higher reconstruction accuracy than other representations. This is supported by experimental results given in Section VI, and simulation study reported by other researchers [2]. Because of high reconstruction accuracy, the PLA retains local patterns and evolution trends better than other representations. In summary, a properly obtained PLA representation along with segment features has the potential to satisfy 6 out of 7 requirements identified in Section II.

#### IV. THE HIERARCHICALPIECEWISE LINEAR REPRESENTATION

The HPLA is a multi-level representation of time series data which facilitates the development of effective and efficient algorithms for coarse-fine processing and mining of time series data. The representation is developed to permit the determination of similarity between two time series when they share similar subsequences, patterns, or time ordered sequence of patterns. It is effective in handling differences in length, translation, time and amplitude scales, minor warp, and even some missing data. It is also possible to determine similarity between two time series using mathematical distance measures (quantitative) or general shape (subjective). The approach, by treating time series as a curve in the time-amplitude binary image, takes advantage of the well-established chaincode based curve smoothing and segmentation methods in the area of image processing [12], [13]. A step-by-step description of obtaining the HPLA representation of a time series is given below.

*Step 1: Normalize the time series.*

The time series  $X = \{x_1, x_2, \dots, x_n\}$  is normalized by replacing amplitude  $x_i$  by  $(x_i - m)/\sigma$ , for  $1 \leq i \leq n$ , where  $m$  and  $\sigma$  are the mean and standard deviation of all amplitude values of the time series.

*Step 2: Digitize the normalized time series and obtain the chaincode representation of the resulting curve.*

In digital image processing, a curve is often represented compactly by its chaincode [12]. The chaincode of a curve is simply a sequence of directional codes, where the  $i^{th}$  element of the chaincode specifies the direction of the  $i^{th}$  pixel (point) relative to the  $(i-1)^{th}$  pixel along the curve. A 3-bit binary code is used to encode the 8 possible directions.

The time series  $X$  may be considered as an open curve in the time-amplitude image space. As time increases monotonically, if  $X$  is represented as a digital curve by digitizing its values, from any pixel on the curve the next pixel can be reached by moving one unit in one of the five possible directions shown in Fig. 2. The algorithm for obtaining the chaincode of the time series  $X$  without actually transforming  $X$  to a digital image is given below.

```

Generate_Chaincode ( $X, n, b$ )
//  $X$ : input time series of length  $n$ 
//  $b$ : amplitude resolution for quantizing elements of  $X$ 
Chaincode  $\leftarrow$  empty list
 $p = \text{int}(x_1/b + 0.5)$ ;  $i = 2$ ;
while ( $i \leq n$ )
     $q = \text{int}(x_i/b + 0.5)$ ;
    if ( $q = p$ )
        Append 2 to Chaincode;
    else if ( $q > p$ )
        Append 3 followed by  $(q-p-1)$  4s to Chaincode
    else
        Append 1 followed by  $(q-p-1)$  0s to Chaincode
     $p = q$ ;  $i++$ ;
return Chaincode
    
```

The above algorithm digitizes the time series, and the generates the chaincode in one pass in linear time ( $O(n)$ ). The digital curve shown in Fig. 1 is obtained by digitizing the normalized amplitude values of a time series of length 77 with a bin size of 0.1. The chaincode of the curve is  $\{243322334334334334223222011011111321210010100110134343444344434443444334334433433231133100010010010010010001011221011212\}$ . Note, the length of the chaincode is greater than the length of the time series due to filling. As the chaincode is computed using sliding window approach and discarded after the computation of feature vector, space is not a major issue.

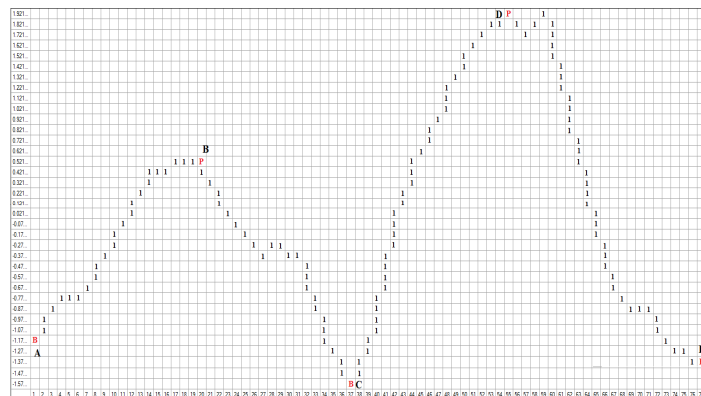


Figure 1. Digitized time series with break points.

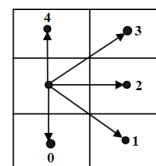


Figure 2. Directional codes.

*Step 3: Determine perceptually important maxima and minima of the curve.*

Nabors has defined four types of curves - type 1, type 2, type 3, and type 4 [12]. The slope along a type 1 curve is between negative infinity and -1, and is represented by a sequence of direction codes 0 and 1. The slope along a type 2 curve is between -1 and 0, and is represented by a sequence of direction codes 1 and 2. The slope along a type 3 curve is between 0 and 1, and is represented by a sequence of direction codes 2 and 3. Finally, the slope along a type 4 curve is between 1 and infinity, and is represented by a sequence of direction codes 3 and 4.

The chaincode of the curve is partitioned into non-overlapping subsequences, where each subsequence represents one of the four curve types. Points at which type 1 or type 2 curves meet type 3 or type 4 curves are local maxima or minima. For a local minimum (maximum), a type 1 or type 2 (type 3 or type 4) curve is followed by a type 3 or type 4 (type 1 or type 2) curve. Instead of selecting all, only the prominent local maxima and minima are selected as break points. A local maximum is taken as a prominent maximum if its raise from the immediately preceding



minimum is greater than the average of raises of all maxima. The algorithm is given below.

```

DetermineProminent_MaxMin(ChainCode)
(MaxMin, MaxMinIndex) = FindMax&Min(ChainCode);
(avgRaise, avgFall) =
FindAverageRaise&Fall(MaxMin);

for i=0 to length(MaxMin)-1
  if MaxMin(i) > avgRaise OR MaxMin(i) < avgFall
    Prominent_MaxMin_I.add(MaxMin(i));

Prominent_MaxMin_Index_I.add(MaxMinIndex(i));
i=0 ;
while i < length(Prominent_MaxMin_I - 1)
  if Prominent_MaxMin_I(i) > 0
    Add index of the global maximum between
    Prominent_MaxMin_Index_I(i) and
    Prominent_MaxMin_Index_I(i+1)
    (both inclusive) to Prominent_MaxMin_Index_F;

  if Prominent_MaxMin_I(i) < 0
    Add index of the global minimum between
    Prominent_MaxMin_Index_I(i) and
    Prominent_MaxMin_Index_I(i+1)
    (both inclusive) to Prominent_MaxMin_Index_F;
    
```

The function FindMaxMin finds all local maxima and minima, and stores their values and indices in MaxMin and MaxMinIndex, respectively. The average raise and fall in value between adjacent maximum and minimum are computed by FindAverageRaise&Fall. Each local maximum with a raise from its immediately preceding minimum greater than average raise becomes an initial prominent maximum. The initial prominent minima are selected, similarly. The initial list of prominent maxima and minima is refined such that maxima and minima appear alternately in the final list. The algorithm identifies two local maxima and three minima (including two end points) as break points for the curve in Fig. 1. These break points are labeled A, B, C, D, and E.

*Step 4: Smooth the curve segments between adjacent break points.*

The curve segment connecting adjacent break points is smoothed by directly modifying the chaincode. A smoothing algorithm similar to the algorithm given by Kim is used for this purpose [13]. Unlike Kim’s algorithm which first requires the identification of distorted sections of the curve, the new algorithm operates on the entire chaincode and selectively modifies elements likely responsible for distortion. It has been shown that the chaincode based algorithm keeps most of the points in their original positions as it smoothes the curve. The smoothing suppresses minor fluctuation due to noise, and is usually reduces the number of partitions into which the segment is partitioned in *Step 5*.

*Step 5: Recursively partition each curve segment.*

Each smoothed curve segment between adjacent break points is partitioned into two sub-segments, and each sub-

segment is represented by the line joining its endpoints. If the mean square error or representation error (average of the square of the vertical distances between the approximating line and points on the curve) between a sub-segment and its approximating line is greater than a pre-specified tolerance  $\epsilon$  then the sub-segment is partitioned again into two parts. Otherwise, it is not partitioned further. This recursive process continues until representation error becomes less than  $\epsilon$  for all sub-segments. A curve segment may be partitioned at its midpoint or best point. The best point is defined as the point that minimizes the sum of the representation errors of the two sub-segments. The resulting HPLA of each segment is represented by a binary tree. The HPLA partitioning of the time series in Fig. 1 is shown in Fig. 3. It is obtained by recursively partitioning curve segments at midpoint until the mean square error between the curve and the approximating line becomes less than 0.5.

*Step 6: Compute feature vectors.*

In the HPLA representation, curve segment between adjacent break points is represented by a binary tree. Each non-leaf node of the binary tree represents a part of the curve segment, and its child nodes represent its two partitions. Let,  $length-l$  and  $slope-l$  denote the length and slope of the line approximating the left partition, and  $error-l$  denote the root mean square error of the left partition. Similarly, length, slope and error of right partition are  $length-r$ ,  $slope-r$ , and  $error-r$ . In this paper, the features used are  $length-l/length-r$ ,  $slope-l/slope-r$  and  $error-l/error-r$ . Other features describing relative shape of the two curve segments and proximity of each curve segment to its approximating line may be used. The feature vector of a leaf-node specifies the segment’s endpoints.

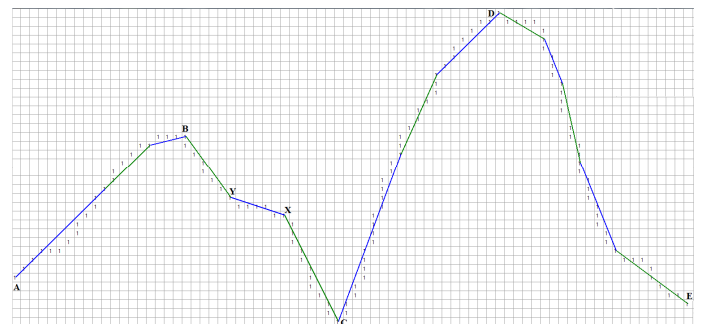


Figure 3. Segmentation of time series in figure 1.

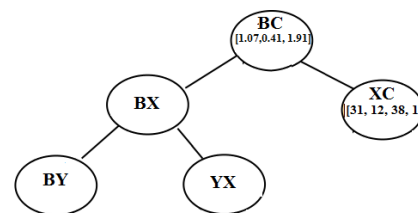


Figure 4. The HPLA representation of segment BC of the time series.

For the purpose of illustrating the computation of feature vectors, consider the binary tree of the segment BC in Fig. 4. The root node represents BC, and its child nodes represent

BX and XC, where X is the mid-point of the chaincode of BC. The length and slope of line BX are 14.21 and -0.82, respectively. The root mean square error between curve BX and line BX is 0.56. Similarly, the length and slope of line XC are 13.41 and -2.0, respectively. The root mean square error between curve XC and line XC is 0.47. Therefore, the 3X1 feature vector of the root node is [1.07 0.41 1.91]. It is also possible to compute features invariant to amplitude scale, amplitude shift, and time scale. The 4X1 feature vector of the leaf-node representing curve XC is [31 12 38 1].

## V. ADVANTAGES OF HPLA REPRESENTATION

Many benefits of the HPLA representation are explained with the help of an example. Consider the task of determining the similarity between two time series  $T$  and  $Q$ . In the simplest case,  $T$  and  $Q$  are of same length, and there is one-to-one correspondence between their elements. Then almost all representations are able to compute a meaningful measure of similarity between  $T$  and  $Q$  in the representation space. This is not true if  $T$  and  $Q$  are unequal in length and their elements do not align. Now, assume that  $T$  and  $Q$  differ in length and their elements do not align due to translation or difference in temporal scale. The goal is to find  $T[a: b]$  and  $Q[c: d]$ , the largest subsequences of  $T$  and  $Q$  that are similar to each other. Larger the length of the subsequences greater is the similarity between the two time series. When the non-alignment is only due to translation,  $T_{a+i}$  aligns with  $Q_{c+i}$ . If the temporal scales of  $T$  and  $Q$  are different then  $T[a: b]$  and  $Q[c: d]$  are similar in shape. However, establishing one-to-one correspondence between elements of  $T$  and  $Q$  is not possible.

The HPLA representation preserves prominent local maxima and minima as break-points, and represents the subsequence between adjacent break-points by a binary tree. The feature vectors of the non-leaf nodes of the binary tree can be invariant to time/amplitude translation and scale. Therefore, it is possible to determine possible correspondence between break-points in  $T$  and  $Q$ . Then a binary tree matching algorithm may be used for the identification of the longest sequence of binary trees in the HPLA representation of  $T$  that matches a sequence of binary trees in the HPLA representation of  $Q$ .

The HPLA representation permits the user to choose coarse or fine approximation depending on the level of accuracy needed, and is natural for coarse-fine processing of time series data. The ability to determine similarity by matching individual sections allows flexibility in defining similarity, and supports the development of section based clustering, classification and indexing methods.

## VI. EXPERIMENTAL RESULTS

Eleven different data sets (7 data sets from UCR archive [15], one from UC Irvine KDD archive, and 3 stock market data sets) are used in the comparative study. From each data set, 10 time series are selected randomly, and the reconstruction error for the HPLA representation is computed for each of them as described below.

1) The time series is normalized to have zero mean and unit standard deviation.

2) The normalized time series is transformed into a digital curve by digitizing the amplitude values with a bin-size of 0.01.

3) The curve is partitioned into segments by choosing perceptually prominent maxima and minima as break points.

4) The HPLA representation is obtained by recursively partitioning each segment at the best point ( $\epsilon = 5$  in pixels or 0.05 in original values).

5) Using the information in root nodes of segments, an approximation of the time series is reconstructed. The compression ratio (percent) and the mean square error between the original time series and the approximation are calculated.

6) An approximation better than the one in 5 is constructed by using the information in root nodes and their non-leaf child nodes. The compression ratio and reconstruction error for this case are also calculated.

TABLE II. EXPERIMENTAL RESULTS

Data Set	Compression Ratio	Reconstruction Error		
		HPLA	PAA	PLA
Mallat	95.4	0.01557	0.13176	0.06825
	92.6	0.01331	0.06130	0.02255
Pseudo Periodic Synthetic	97.2	0.02205	0.08922	0.04603
	94.6	0.00051	0.03560	0.00962
OliveOil	93.8	0.01401	0.11397	0.08470
	90.2	0.00251	0.05550	0.03365
Adiac	88.5	0.00359	0.03193	0.01131
	84.7	0.00339	0.00894	0.00465
Yoga	91.3	0.00451	0.01836	0.00646
	87.7	0.00132	0.00748	0.00184
Fish	93.9	0.00413	0.09535	0.05453
	90.1	0.00235	0.08044	0.00402
Swedish Leaf	86.8	0.02613	0.13432	0.06100
	80.3	0.02235	0.03474	0.02465
OSU Leaf	92.5	0.02119	0.07163	0.03406
	88.3	0.00893	0.04130	0.01648
Amazon	90.7	0.01898	0.08219	0.02832
	86.4	0.00321	0.03154	0.00552
IBM	87.4	0.03515	0.12166	0.05139
	80.8	0.02199	0.09529	0.03357
Microsoft	83.5	0.03185	0.10156	0.03763
	79.6	0.01811	0.06188	0.02157

The average compression ratio and reconstruction accuracy for each set given in Table II. For each set there are two entries. The first entry is coarse (step 5), and the second entry is relatively finer than the first entry (step 6). The PAA and PLA representations of each time series are obtained by partitioning time series into equal length segments. Each PLA segment is fitted with the best line using linear regression. The number of segments is adjusted for each representation to achieve the same level of compression as the corresponding HPLA representation. As

expected, the HPLA representation achieved significantly higher reconstruction accuracy for all data sets.

## VII. CONCLUSION AND FUTURE WORK

This paper has made two primary contributions. First, seven requirements, a good time series representation should satisfy are explicitly identified by analyzing the needs of time series data mining applications. Secondly, a new time series representation (HPLA), which satisfies six of the seven requirements better than PAA, APCA and PLA representations, is proposed. The distance measure that satisfies the lower bound criterion is not known for the HPLA representation.

The experimental results illustrate, for a given compression ratio, the HPLA represents the time series more accurately than the PAA and uniformly segmented PLA representations. A time series can have many PLA representations based on how it is segmented. The representation accuracy, usefulness, and effectiveness for mining time series is determined by the number of break points, and how well the break points are selected during segmentation. The strength of the HPLA representation comes from the novel two-stage segmentation approach, which identifies the perceptually important local maxima and minima as primary break points. These break points provide a broad perception of the shape. They also identify trend changes. Additional break points are placed between primary breakpoints to achieve the desired degree of accuracy.

The HPLA being a multi-level representation, permits coarse-fine processing of time series. Most time series representations do not (effectively) support the finding the longest subsequence of one time series that has a matching (similar) subsequence in the other time series. The problem becomes even more challenging if the two time series do not have the same time and amplitude scale. The HPLA facilitates aligning corresponding segments of the two time series by using perceptually important primary break points as anchor points. The feature vector, which specifies the relative values of slope, length and error of the two partitions of each segment, is invariant to time and amplitude scale. These two features make the HPLA more suitable than other piecewise linear or constant representations for time series matching.

The clustering, classification, and query by content require a representation that facilitates the development of efficient and effective algorithms to determine the similarity between time series. The preliminary research and limited simulation results suggest that the HPLA representation is highly suitable for almost all time series data mining applications including clustering, classification and query by content. Therefore, future research should focus on the development of the HPLA based algorithms for aligning two time series, matching time series, and clustering and classification of time series based on piecewise matching.

## REFERENCES

- [1] P. Esling and C. Agon, "Time-series data mining," *ACM Computing Surveys*, vol. 45, Dec. 2012, 34 pages.
- [2] E. Keogh, K. Chakrabarti, M. Pazzani and S. Mehrotra, "Dimensionality reduction for fast similarity search in large time series databases," *Knowledge and Information Systems*, vol. 3, pp. 263--286, 2001.
- [3] N. Q. Hung and D. T. Anh, "An improvement of PAA for dimensionality reduction in large time series databases," *Proceedings of the 10th Pacific Rim International Conference on Artificial Intelligence: Trends in Artificial Intelligence (PRICAI '08)*, Jan. 2008, pp. 698-707.
- [4] J. Lin, E. Keogh, L. Wei, and S. Lonardi, "Experiencing SAX: a novel symbolic representation of time series," *Data Mining and Knowledge Discovery*, Oct. 2007, pp. 107-144.
- [5] M. Azzouzi and I. T. Nabney, "Analysing time series structure with hidden Markov models," *Neural Networks for Signal Processing VIII, Proceedings of the 1998 IEEE Signal Processing Society Workshop*, 1998, pp. 402-408.
- [6] F. Korn, H. V. Jagadish, and C. Faloutsos, "Efficiently supporting ad hoc queries in large datasets of time sequences," *Proc. ACM SIGMOD international conference on Management of data*, 1997, pp. 289-300.
- [7] E. Keogh and M. Pazzani, "An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback," *Fourth International Conference on Knowledge Discovery and Data Mining (KDD'98)*, 1998, pp. 239-241.
- [8] C. Yan, J. Fang, L. Wu, and S. Ma, "An approach of time series piecewise linear representation based on local maximum, minimum and extremum", *Journal of Information & Computational Science*, June 2013, pp. 2747-2756.
- [9] K. B. Prat and E. Fink, "Search for patterns in compressed time series [J]," *International Journal of Image and Graphics*, vol. 2, Issue. 1, 2002, pp. 89-106.
- [10] S. Park, S. W. Kim, and W. W. Chu, "Segment-based approach for subsequence searches in sequence databases," In *Proceedings ACM symposium on Applied computing (SAC '01)*, 2001, pp. 248-252.
- [11] J. Zhou, G. Ye, D. Yu, "A new method for piecewise linear representation of time series data," *Physics Procedia*, vol. 25, 2012, pp. 1097-1103.
- [12] D. H. Nabors, A boundary based image segmentation and representation method for binary images, *Doctoral Dissertation*, The University of Alabama in Huntsville, 2000.
- [13] S. K. Kim, Hierarchical representation of edge images for geometric feature based image interpretation, *Doctoral Dissertation*, The University of Alabama in Huntsville, 2007.
- [14] K. Chakrabarti, E. Keogh, S. Mehrotra, and M. Pazzani, "Locally adaptive dimensionality reduction for indexing large time series databases", *ACM Transaction on Database Systems*. vol. 27, Jun. 2002, pp. 188-228.
- [15] E. Keogh, Q. Zhu, B. Hu, Y. Hao, X. Xi, L. Wei and C. A. Ratanamahatana (2011). *The UCR Time Series Classification/Clustering*. [retrieved: August, 2013] Homepage: [www.cs.ucr.edu/~eamonn/time\\_series\\_data/](http://www.cs.ucr.edu/~eamonn/time_series_data/)

# Cache Management for Aggregates in Columnar In-Memory Databases

Stephan Müller, Ralf Diestelkämper, Hasso Plattner

Hasso Plattner Institute

University of Potsdam

Potsdam, Germany

Email: {stephan.mueller, ralf.diestelkaemper, hasso.plattner}@hpi.uni-potsdam.de

**Abstract**—Modern enterprise applications generate workloads of short-running transactional queries as well as long-running analytical queries. In order to improve the execution time of computationally intensive analytical queries we have introduced an aggregate cache that makes use of the typical main-delta architecture of columnar in-memory databases (IMDBs) to cope with data modifications. In this work, we contribute a cache management system for that aggregate cache, which bases the cache admission and replacement decisions on novel profit metrics. These metrics are tailored to the main-delta architecture of IMDBs. They ensure that expensive aggregates are stored in the cache while light weight query results are rejected. For the profit estimation of a cached aggregate the system also takes into account transactional data modifications triggered by the enterprise application. Along with the profit metrics we introduce an asynchronous cache management algorithm designed for the main-delta architecture as well as the transactional data modifications. We evaluate the cache management system on mixed, transactional and analytical workloads and real customer data.

**Keywords**—Aggregates, Materialized Views, Cache Management, In-Memory Database, Column Store

## I. INTRODUCTION

In the past, transactional and analytical queries have been associated with two separate applications for transactional processing (OLTP) or analytical processing (OLAP). This distinction is no longer applicable for modern enterprise applications [1], [2] because they make use of both, online transactional and analytical queries. In a contemporary financials and controlling application typical OLTP-style queries insert new bookings, whereas OLAP-style queries aggregate the records for profit and loss statements.

The OLAP-style queries may take a significant time to be processed [1]. That is why we have developed an aggregate cache which leverages the main-delta architecture of columnar in-memory databases in order to speed up recurring analytical queries in a consolidated environment [3], [4]. In a columnar IMDB, a table is stored by column vectors instead of row tuples and resides in memory. Since inserts of new tuples are usually more expensive in a column store than in a row store database, each table has a highly-compressed, read-optimized main storage and a write-optimized delta storage. The delta storage persists the transactional manipulations made to the database table. It contains orders of magnitude less tuples than the main storage. However, when the delta storage reaches a certain threshold, a merge process is triggered [5] that merges the tuples of the delta storage into the main storage.

Our aggregate cache stores the aggregation result returned

from the main storage when an analytical query is executed for the first time. In the following, we call the result aggregate. If the same query is executed again, the cached aggregate is combined with the on-the-fly calculated result on the delta storage and returned to the application. With this approach the aggregate cache can provide an up-to-date result and save significant computation and execution time overhead, because the delta storage is much smaller than the main storage and on-the-fly aggregations on it are relatively fast.

In this work, we present a cache management system for the aggregate cache. It ensures that the aggregate cache does not grow arbitrarily large. Additionally, it prevents the aggregate cache from keeping unused or computationally lightweight aggregates in the cache. For the identification of such aggregates, the cache management system makes use of a profit metric. The metric assesses the performance benefit obtainable from each aggregate if it remains in the cache. Existing profit metrics are calculated from multiple runtime metrics such as the access rate of a cached aggregate, the execution time to calculate the aggregate, and the aggregate's size. These metrics are not optimal for the aggregate cache because they do not distinguish between the calculation time on the main storage and calculation time on the delta storage. However, that is important for the aggregate cache since it cannot accelerate recurring queries whose calculation time mostly originates in the on-the-fly aggregation on the delta storage. Thus, novel profit metrics are required for the aggregate cache that consider the main-delta architecture and the mixed workload.

The aggregate cache is designed to concurrently handle the database requests from multiple users and applications. That is why the management system should avoid blocking behavior during query processing. Previous cache management systems performed synchronous cache management [6]–[8]. The synchronous management has caused blocking behavior for every processed query. That is why we introduce an asynchronous cache management algorithm, which evicts cached aggregates decoupled from the query processing.

During the merge phase of a base table, the current implementation of the aggregate cache removes those aggregates from the cache whose base table is merged. The consequence is that the aggregate cache has to recalculate the aggregates from scratch the next time they are required. Instead of evicting the aggregates, a sophisticated cache management system can incrementally revalidate the affected aggregates while the merge process is in progress. The incremental revalidation process is basically the same process that the cache manager performs

on a cache hit, except that the aggregate cache updates the cache entry after it has been combined with the result from the on-the-fly aggregation on the delta storage. For many cached aggregates, the on-the-fly aggregation is lightweight compared to the complete recalculation. Thus, the cache performance can be significantly improved if cached aggregates are revalidated instead of evicted. That is why we describe an aggregate revalidation algorithm for the merge process.

Instead of using a mixed workload benchmark like the CH-Benchmark [9], we evaluate our cache management system on a financials and controlling scenario that is based on real world data and query templates from an operational enterprise system. For the evaluation purpose, we have implemented the cache management system in SanssouciDB, a columnar IMDB with main-delta architecture.

In the following section, we describe how our cache management system differs from other existing cache management systems. In Section 3, we give an architectural overview of the aggregate cache with our cache management extensions. We introduce the novel profit metrics in Section 4 and the asynchronous cache management algorithm in Section 5. In Section 6 we describe the revalidation algorithm for the merge process. We evaluate the profit metrics and the algorithms on the financials and controlling scenario in Section 7. In Section 8, we summarize our results and give an outlook on future work.

## II. RELATED WORK

Caches are not only applied in database systems but also in systems for disk buffering or client-server applications. Each of these systems requires a cache management system to maintain the cache.

In disk buffering systems, the cache is used to provide fast access to data on disk. Disk buffering systems have to manage a limited size of cache space. Hence, they favor frequently and recently accessed data blocks over rarely touched blocks in order to maximize the system's performance. In the past, many algorithms have been proposed to most profitably manage the cache. Least-frequently-used (LFU) [10], least-recently-used (LRU) [11], k-least-recently-used (LRU-K) [12], 2Q [13], MultiQueue [14] and least-recently-frequently-used (LRFU) [15] are just a few to be mentioned. It takes only minimal effort to adjust these algorithms to work with the aggregate cache. However, they do not consider the cached aggregate's size, the execution time for a cache hit and the execution time for a cache miss. These parameters are assumed to be equal or at least almost equal in a disk buffering system, but can significantly differ in a cache system for database aggregates. For example, we have two analytical queries which have been executed similarly frequently in the recent past. The first analytical query has a processing time of several seconds when it is not cached, but runs only a couple milliseconds when it is cached. The second aggregate query may take only several hundred milliseconds overall execution time when it is not cached and about the same time when it is cached. The above mentioned algorithms would not prefer one query over the other. Our cache management system should prefer the first query over the second in order to maximize the saved processing time.

Another popular application for caches and, hence, cache management systems lies in client-server systems. A client retains information received from the server via a network in

order to avoid redundant data transfer and network contention. Client-side caches have also been introduced for database systems [16]. Opposite to these caches our aggregate cache resides on the server. The server side cache can serve multiple tenants working on a single consolidated database system. In this way, multiple clients can profit from a single cached aggregate on the server.

A couple of cache management systems for materialized views and query results have already been introduced and implemented in the past. In the following, we first provide a non-comprehensive overview over previous cache management systems and then distinguish them from our aggregate cache.

Scheuermann et al. introduce WATCHMAN [6] system as one of the first approaches to manage database query results. It makes cache admission and replacement decisions based on the execution frequency, the execution time, and the result set size of a query. The authors show that WATCHMAN significantly improves the cache performance over an at that time sophisticated disk buffer management algorithm LRU-K. In a follow-up project the group around Scheuermann extended the original WATCHMAN system described in [6] to support subqueries and to consider update costs for the result sets [17].

Kotidis et al. implement a view management system called DynaMat for data warehouses. It dynamically manages materialized aggregates [7]. The authors evaluate four metrics for cache admission and replacement, the frequency metric, the execution time metric, the result set size metric and a combination of the three of the previous metrics. They show that the combined metric, which is similar to the WATCHMAN metric, performs best.

Park et al. design a caching mechanism for OLAP systems, which is able to reuse partial results for related queries in drill-down and roll-up sequences [8]. The cache admission and replacement algorithm is extended in order to take into account the profit of a query result for multiple related queries. Related queries are part of the same drill-down or roll-up sequences and were either executed in the recent past or will very likely be executed in the near future. The reuse of partial results saves their system expensive random accesses to the disk. Therefore, the system performance can significantly profit from reusing partial and overlapping query results residing in memory. In our setup, however, the data is already stored in memory and random accesses to disk are no limiting factor any more. Additionally, Park et al. are limited to matching canonical drill-down and roll-up sequences. Real world enterprise workloads contain more complex analytical queries with subqueries and joins. That is why, in the recent years, more sophisticated research has been done on reusing partial views mitigating a canonical query schema [18]. However, this is out of scope of this work.

Opposite to all the above cache management systems our management system administers an aggregate cache on an IMDB with main-delta architecture. It considers the costs to do an on-the-fly aggregation on the delta storage for every query answered from the cache. That was unnecessary in the above system setups, because the cached aggregate was delivered as is.

Additionally, none of the above systems are designed for mixed workloads in which data modifications can occur at any time. Some of them consider bulk data modifications during dedicated maintenance intervals but they cannot process combined online and analytical workloads. In contrast, our

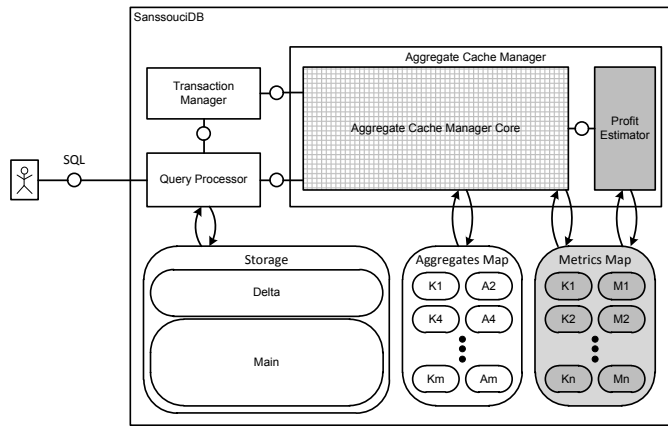


Figure 1. The Aggregate Cache Manager inside SanssouciDB

cache management system is designed to deal with mixed workloads.

### III. AGGREGATE CACHE ARCHITECTURE

The aggregate cache and the cache management system are fully integrated into SanssouciDB. Figure 1 gives an architectural overview over the cache management system. The white components are already existing components, while the gray parts are cache management extensions that are implemented for this work. The textured gray component is extended in this work in order to support cache management. Figure 1 shows that the aggregate cache consists of three major components, the aggregate cache manager, the aggregates map, and the metrics map.

The aggregate cache manager consists of the cache manager core and the profit estimator. The cache manager core stores the cached aggregates in the aggregate map and updates the runtime information in the metrics map. It receives cacheable aggregates from the query processor and delivers cached aggregates back to the query processor if a query can be answered from the cache. Before a cached aggregate is delivered to the query processor, the cache manager core checks whether the cached aggregate is still up to date. It asks the transaction manager if rows were invalidated in the main storage since the point in time the aggregate was created. In case rows were invalidated, the aggregate cache updates the cached aggregate before it delivers the aggregate to the query processor.

The profit estimator is an extension to the existing aggregate cache. It calculates the profit for every cached aggregate by applying a profit metric on the runtime information residing in the metrics map.

### IV. PROFIT METRICS

The better the profit metric assesses the benefit of a cached aggregate, the better the aggregate cache can perform. A higher cache performance, in turn, leads to a better system performance. Before we introduce the novel metrics, we assess existing profit metrics in buffer management systems and previous query result caches in the context of the aggregate cache.

#### A. Existing Buffer Profit Metrics

We start with the metrics originating in disk buffering systems. The symbols used to describe the following metrics

are explained in Table I:

- 1) **LRU**: The LRU metric defines the profit of cached aggregates by their last access [11].

$$profit_{LRU}(q) = \frac{1}{t - last\_access_q} \quad (1)$$

- 2) **LRU-K**: An extension of the LRU metric is the k-recently used metric which considers the k most recent accesses of a cached result [12].

$$profit_{LRU-K}(q) = \begin{cases} \frac{1}{t - kth\_access_q}, & \text{if } \geq k \text{ accesses} \\ 0, & \text{else} \end{cases} \quad (2)$$

- 3) **LFU**: The least frequently used metric (LFU) rates the cached aggregates by their recurrence. The recurrence increases constantly with every access [10].

$$profit_{LFU}(q) = recurrence_q \quad (3)$$

- 4) **LRFU**: The least frequently recently used metric (LRFU) rates aggregates by their recurrence and recency of access. It actually is not a single metric, but a spectrum of metrics covering the range between the LRU and the LFU metric [15].

$$profit_{LRFU}(q) = \left(\frac{1}{2}\right)^{\lambda \cdot (t - last\_access_q)} \quad (4)$$

with  $0 \leq \lambda \leq 1$

For  $\lambda = 0$   $profit_{LRFU}(q) = profit_{LFU}(q)$  and for  $\lambda = 1$   $profit_{LRFU}(q) = profit_{LRU}(q)$ . This is proven in [15]. If  $\lambda$  lies in between 0 and 1 the LRFU metric returns a profit between the LFU profit and the LRU profit. Such a profit is desirable, because the recurrence and the recency with which a cached aggregate is accessed are two important indicators for the benefit of a cached aggregate.

The above metrics only consider access rates and frequencies of cached aggregates. This is sufficient for disk buffering systems, because they manage disk blocks of equal size and similar disk fetch times. In database result caches or aggregate caches like ours, these metrics do not perform well, because the aggregate sizes can differ significantly. Some may contain only a couple groups and only a few columns while others have hundreds or even thousands of groups and several dozens to hundreds columns. Additionally, the time needed to calculate the query result or the cached aggregate can differ substantially. Some calculations finish after a few milliseconds while others process for several seconds or even longer.

#### B. Existing Query Cache Profit Metrics

The profit metrics of previous query result caches reflect the above thoughts, since they consider the result set size as well as the execution time of the cached query.

- 1) **WATCHMAN**: The WATCHMAN metric considers the k last references as well as the aggregate size and the execution time of a query [6].

$$profit_{WATCHMAN}(q) = profit_{LRU-K}(q) \cdot \frac{t_q}{result\_size_q} \quad (5)$$

- 2) **DynaMat**: Similarly to the WATCHMAN metric, the DynaMat metric takes the result size and the

TABLE I. DEFINITION OF SYMBOLS

Symbol	Definition
$t$	current time
$last\_access_q$	time of the last access to aggregate $q$
$kth\_access_q$	time of the $k$ th last access to aggregate $q$
$recurrence_q$	number of times aggregate $q$ has been referenced
$t_q$	calculation time for an aggregate $q$ on the main and delta storage without cache
$t_{main}$	calculation time for an aggregate $q$ only on the main storage
$t_{cache+\Delta}$	calculation time for an aggregate $q$ on the cache and the delta storage
$result\_size_q$	memory space required to store an aggregate $q$ that is calculated on the main storage
$delta_q$	number of aggregated tuples in the delta storage relevant for aggregate $q$
$main_q$	number of aggregated tuples in the main storage relevant for aggregate $q$
$inval_q$	number of invalidated tuples in the main storage affecting aggregate $q$
$\Delta_q$	number of tuples involved in the delta compensation

calculation time of an aggregate into account. They are combined with the recurrence of the query [17].

$$profit_{DynaMat}(q) = profit_{LFU}(q) \cdot \frac{t_q}{result\_size_q} \quad (6)$$

These metrics worked well in previous query result caching systems [6], [7]. However, the aggregate cache differs in at least two substantial characteristics from the previous caches, which should be reflected in the profit metric. First, the cache is tailored to the main-delta architecture of columnar IMDBs. Second, the cache has to deal with invalidated tuples on the main storage.

#### C. Novel Profit Metrics for the Aggregate Cache

The above metrics only consider the overall query execution time. They do not distinguish between processing an aggregation on the main storage and on the delta storage. The differentiation is important for the performance of the cache as the following example demonstrates. The aggregate cache keeps only the aggregation result from the main storage and performs an on-the-fly aggregation on the delta storage for every incoming query. If a query only touches tuples in the delta storage and the overall execution time of the query is considerably long, the cache does not speed up this query. However, the above metrics assign the associated aggregate a high profit, because the overall execution time is long and the cached (empty) aggregate has a small size.

The following four metric extensions are tailored to the aggregate cache architecture. They distinguish between the aggregation on the main storage and the delta storage.

- 1) **TAR:** The tuples aggregated ratio (TAR) metric represents the ratio of the tuples aggregated on the main storage and the tuples aggregated on the delta storage for a query  $q$ . The ratio yields more profitable results, if the number of aggregated delta tuples is low or the count of processed main tuples is high. That is desirable, because the tuples on the delta are aggregated whenever  $q$  is processed, whereas the tuples on the main are aggregated only when the aggregate is cached.

$$profit_{TAR}(q) = \frac{main_q}{\Delta_q + 1} \quad (7)$$

Note that a "+1" is added to the tuples touched in the delta storage in order to avoid a zero division. For the simple case of just having a single table,  $\Delta_q = delta_q$ .

- 2) **ETR:** The execution time ratio (ETR) metric is the proportion the main storage processing time and the delta storage processing time of a query  $q$ . It favors queries that have a short processing time on the delta storage and a long processing time on the main storage. We want to cache the aggregates of these queries, because they are answered quickly from the cache, but they need a significant amount of time if they are calculated from the main storage.

$$profit_{ETR}(q) = \frac{t_{main}}{t_{cache+\Delta}} \quad (8)$$

- 3) **TAD:** The tuples aggregated difference (TAD) metric considers the number of tuples that do not have to be aggregated when a query  $q$  is answered with the help of the aggregate cache. The tuples on the delta storage are aggregated on a cache hit and miss to answer  $q$ . The tuples on the main storage are not aggregated on a cache hit, because the aggregation result is cached by the aggregate cache. Their count is the number of tuples saved on a cache hit. The higher it is, the higher is the profit of  $q$ 's aggregate.

$$profit_{TAD}(q) = (main_q + \Delta_q) - \Delta_q = main_q \quad (9)$$

- 4) **ETD:** The execution time difference (ETD) metric reflects the time saved when a query  $q$  is executed with the aggregate cache. The processing time on the delta storage is needed to answer  $q$  in both cases, when the  $q$ 's aggregate is cached and when it is not cached. Therefore, the time saved when  $q$ 's aggregate is cached is the main processing time. The higher this main processing time is, the higher is the profit of  $q$ 's aggregate.

$$profit_{ETD}(q) = (t_{main} + t_{cache+\Delta}) - t_{cache+\Delta} = t_{main} \quad (10)$$

Manipulative database transactions like deletes and updates can invalidate rows in the main storage [19]. Thus, the deletes and updates potentially have an impact on the cached aggregates. In case the deleted rows are part of a cached sum the rows are added up and then subtracted from the cached sum. This on-the-fly process occurs whenever a query is answered from the aggregate cache.

To address the invalidation of rows in the main storage and the resulting compensation process in the cache, we define the following compensation factor that is based on the number

of invalidated tuples relevant for aggregate  $q$ ,  $inval_q$ , and the number of aggregated tuples  $main_q$  in the main storage:

$$icomp(q) = \frac{1}{2} - \frac{inval_q}{main_q} \quad (11)$$

If more than 50% of the aggregated tuples are being invalidated, the profit becomes negative; which indicates that an on-the-fly aggregation on the main storage is more cheaper than using the cached aggregate.

The novel profit metrics combine one of the four main-delta metric extensions with the invalidation compensation extension, the LRFU metric, and the size of the cached aggregate. That allows them to assess the profit of each cached aggregate more precisely than the existing profit metrics.

- 1) **AC-TAR:** The aggregate cache tuples aggregated ratio metric (AC-TAR) is a combination of the LRFU metric (cf. Equation 4), the tuples aggregated ratio metric (cf. Equation 7), the invalidation compensation (cf. Equation 11), and the size of the aggregate associated with query  $q$ .

$$profit_{AC-TAR}(q) = profit_{LRFU}(q) \cdot icomp(q) \cdot \frac{profit_{TAR}(q)}{result\_size_q} \quad (12)$$

- 2) **AC-ETR:** The aggregate cache execution time ratio metric (AC-ETR) combines the LRFU metric (cf. Equation 4), the execution time ratio metric (cf. Equation 8), the invalidation compensation (cf. Equation 11), and the size of the aggregate associated with query  $q$ .

$$profit_{AC-ETR}(q) = profit_{LRFU}(q) \cdot icomp(q) \cdot \frac{profit_{ETR}(q)}{result\_size_q} \quad (13)$$

- 3) **AC-TAD:** The aggregate cache tuples aggregated difference metric (AC-TAD) is a combination of the LRFU metric (cf. Equation 4), the tuples aggregated difference metric (cf. Equation 9), the invalidation compensation (cf. Equation 11), and the size of the aggregate associated with query  $q$ .

$$profit_{AC-TAD}(q) = profit_{LRFU}(q) \cdot icomp(q) \cdot \frac{profit_{TAD}(q)}{result\_size_q} \quad (14)$$

- 4) **AC-ETD:** The aggregate cache execution time difference metric (AC-ETD) is assembled from the LRFU metric (cf. Equation 4), the execution time difference metric (cf. Equation 10), the invalidation compensation (cf. Equation 11), and the size of the aggregate associated with query  $q$ .

$$profit_{AC-ETD}(q) = profit_{LRFU}(q) \cdot icomp(q) \cdot \frac{profit_{ETD}(q)}{result\_size_q} \quad (15)$$

## V. CACHE MANAGEMENT ALGORITHM

The algorithm evicts aggregates from the cache that do not improve the overall system performance. Such aggregates may be empty aggregates or aggregates based on meanwhile invalidated rows. On-the-fly recalculation of these aggregates is cheaper than retaining the aggregates in the cache. The algorithm also has to remove the least profitable aggregates from the cache in case the system is running out of memory. Many analytical queries may be processed in parallel along with even more transactional operations. That is why, the algorithm should not block the system's progress with every incoming analytical query.

For the given reasons, we propose a cache management algorithm which maintains the aggregate cache asynchronously to the query processing. In previous systems the aggregate cache was maintained with every incoming analytical query [6]–[8]. We simply add the queries to the aggregate cache and trigger a maintenance routine to evict the least profitable queries in time intervals and when memory space is running low.

The regular cache maintenance and the aggregates revalidation task during the merge phase requires the algorithm to be split into three parts: The first part updates the cache metrics, the second part evicts queries from the cache, and the third part removes entries from the metrics map.

1) *Cache Metrics Update:* This first part of the algorithm updates the information stored in the metrics map. If the aggregate matching the executed query already exists as an entry in the metrics map, the entry is updated. This process requires a read lock on the metrics map, because another process in the system may concurrently remove entries from the map. If there is no matching entry in the map, the procedure creates a new entry and fills it with the information obtained from the system. That requires a write lock on the metrics map, since other procedures handling other analytical queries concurrently may add new entries to the metrics map. However, the algorithm inserts the entry to a hash map so that the procedure has an average complexity of  $O(1)$  [20]. So in comparison to the other parts of our algorithm, the update is processed very quickly and does not cause noticeable blocking behavior.

2) *Cache Trimming:* Trimming the cache is more complex than updating the metrics for a single aggregate. The procedure is described in Algorithm 1. It is executed periodically and when the system's memory space is low.

First, the procedure iterates over the cache metrics map  $M$  and appends each entry whose associated aggregate is cached in a separate list  $ca$ . For this task it acquires a read lock on  $M$ . This process has a complexity of  $O(n)$ , where  $n$  is the number of entries in the cache metrics map. Therefore, the lock time scales linearly with the number of records in  $M$ . The collection process does not block the update of information in existing entries, because that only requires a read lock on  $M$ . However, it blocks the insertion of new entries into the metrics map, because the insertion of new entries requires an exclusive write lock. The insertion of new entries is only necessary, when a unknown query gets cached. In that case, the query has to be processed on both, the main and the delta storage. That processing time exceeds the time to collect all cached aggregates in most cases so that the blocking behavior is hardly noticeable.

As the next step, the procedure sorts the entries in the list  $ca$



by profit in ascending order. Since most of the profit metrics introduced in Section IV require a timestamp, the a current timestamp is handed to the profit estimator that encapsulates the profit metric. The sorting is the most costly part in the cache trimming procedure. It has a complexity of  $O(n \log(n))$ , where  $n$  is the number of cached entries. However, it does not require any lock, because it works on a temporary list. Thus, it cannot cause any blocking behavior.

Given this sorted list of cached entries the procedure starts removing entries from the aggregate map  $A$ . It fetches entries from the front of the list  $ca$  until the new cache size is reached or the profit of an entry is bigger than zero. Recapture that the profit becomes negative if more than 50% of the rows that the cached aggregate is based on are deleted. The removing of cache entries requires locks on both the aggregate map  $A$  and the metrics map  $M$ . The cache trimming procedure obtains and releases them for the eviction of each entry in order to avoid blocking behavior.

If new aggregates are added to the cache, while the procedure is sorting or removing entries from the list  $ca$ . They are ignored for the current trimming process.

---

#### Algorithm 1 Cache Trimming Procedure

---

**Require:** aggregate\_map  $A$ , metrics\_map  $M$ ,  
current\_cache\_size  $cs$ , target\_cache\_size  $ts$

```

1: procedure TRIM_CACHE( $A, M, cs, ts$ )
2:   cached_aggregates  $ca \leftarrow [ ]$ 
3:    $M.acquire\_read\_lock()$ 
4:   for all metrics_map_entry  $\langle k, v \rangle$  in  $M$  do
5:     if  $v.is\_cached$  then
6:        $ca.append(\langle k, v \rangle)$ 
7:     end if
8:   end for
9:    $M.release\_read\_lock()$ 
10:  timestamp  $t \leftarrow current\_time()$ 
11:  profit_estimator.sort( $ca, t$ )
12:  for all metrics_map_entry  $\langle k, v \rangle$  in  $ca$  do
13:    if profit_estimator.profit( $v$ )  $> 0$  then
14:      if  $cs < ts$  then
15:        break
16:      end if
17:    end if
18:     $A.acquire\_write\_lock()$ 
19:     $M.acquire\_read\_lock()$ 
20:     $cs \leftarrow cs - v.result\_size$ 
21:     $M.invalidate(k)$ 
22:     $A.evict(k)$ 
23:     $M.release\_read\_lock()$ 
24:     $A.release\_write\_lock()$ 
25:  end for
26: end procedure

```

---

3) *Metrics Trimming*: Similarly to the cache entries, the metric entries are trimmed in periodic intervals. For the metrics trimming task, all metric map entries for uncached aggregates are obtained and stored in a list. A read lock on the metrics map is required for this operation. It has a complexity of  $O(n)$ , where  $n$  is the number of entries in the cache metrics map. The read lock may block the insertion of unknown aggregates to the cache, but as described previously, query processing should not noticeably be blocked.

The list of uncached aggregates is sorted by LRU in order to find the metric map entries that have not been used for the longest period of time. They are evicted from the metrics map one after another until the defined threshold is reached. For every entry eviction, a write lock on the metrics map is acquired and released after the entry is removed from the metrics map. This fine grained lock handling avoids blocking behavior.

#### VI. INCREMENTAL REVALIDATION ALGORITHM FOR THE MERGE PROCESS

The incremental revalidation algorithm for the merge process updates those cached entries whose underlying base table is merged. It is integrated with the merge process described in [21]. During the merge prepare phase, the algorithm identifies and collects all cached aggregates that are based on the table being merged. It marks the aggregates to indicate that they need revalidation.

When the merge process is in progress, the algorithm orders the cached aggregates by their profit in descending order, so that the most profitable aggregates are at the beginning of the list. It incrementally revalidates the aggregates beginning from the head of the list until the merge process is finished. Directly after the revalidation the aggregates are marked as revalidated.

When the merge process is committed, the algorithm removes all aggregates which have not been updated from the cache, because they do not represent the aggregation result on the new main storage any more.

#### VII. EVALUATION

For the evaluation, we implemented the algorithms and profit metrics in SanssouciDB [21], an IMDB with main-delta architecture. However, we are confident that our algorithm and metrics yield similar results when implemented in other IMDBs such as SAP HANA [22] or Hyrise [23]. We evaluate our algorithms and metrics with a financial accounting application. Other than mixed workload benchmarks such as the CH-benchmark [9] the application works on a database with real customer data. It also generates a mixed workload with OLTP-style inserts for the creation of accounting documents and OLAP-style queries for the calculation of reports like profit and loss statements. Therefore, the financial accounting application suits our evaluation purposes.

The application's database contains 22 million records in a single, denormalized table. We generated inserts based on these records for our workload. We also extracted 100 OLAP-style aggregate queries from the application and validated these with domain experts. They contain at least one aggregation function. From these 100 distinct OLAP-style queries, we create an analytical workload with 1000 queries, which we use for the following experiments. The server for the benchmarks has 4 Intel Xeon processors with a total of 40 physical cores and 1 TB of main memory.

##### A. Delta Storage Tuples

In the first experiment, we vary the number of tuples in the delta storage and compare the four aggregate cache metrics AC-ETD, AC-TAD, AC-ETR, and AC-TAR with the existing profit metrics LRU, WATCHMAN (WM), and DynaMat (DYN). The results are displayed in Figure 2. As performance measure we use the workload execution time. The lower it is, the better a metric performs. We set the eviction threshold to 80%. As a consequence the cache is trimmed to 80% of its

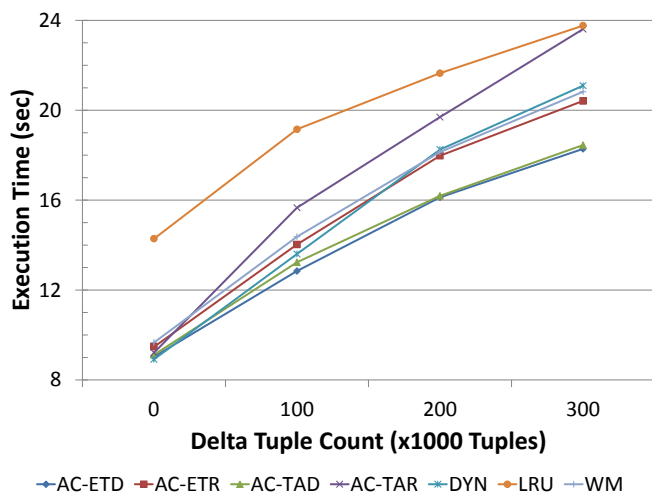


Figure 2. Comparison of Profit Metrics with Varying Delta Sizes. (Cache Size 200kB, Eviction Threshold 80%)

maximum size when the maximum size of 200kB is reached. 200kB are sufficient to cache about 40 to 50 aggregates. The LRFU decay factor  $\lambda$  is set to 0.0001. This value has proven to perform best in our experiments.

The results show that the execution times of all profit metrics increase with a growing delta storage. The reason is that the aggregations on the delta storage become more expensive with an increasing delta size. When the delta storage is empty, all metrics, except the LRU metric, show similar performance results. That is the case, because the delta access time is very low and almost equal for all queries. With increasing delta size, the AC-ETD and AC-TAD metrics more and more outperform the other metrics. When the delta contains 200,000 tuples or more, the two metrics perform at least two seconds better than the existing WATCHMAN and DynaMat metrics. That is a performance gain of at least 11%. The AC-ETD and AC-TAD metrics constantly yield workload execution times that are five to six seconds faster than the LRU metric, independent from the delta size.

The ratio metrics AC-ETR and AC-TAR perform one and a half to six seconds worse than AC-ETD and AC-TAD metrics. The reason is that the ratio metrics assign high profit to all aggregates that consider only few tuples on the main and the delta storage. For example, an aggregate  $a$  is computed over 2 tuples in the delta storage and 100 tuples in the main storage. An aggregate  $b$  is calculated over 2,000 tuples in the delta storage and 50,000 tuples in the main storage. When the size and access history is equal for  $a$  and  $b$ , the AC-TAR metric assigns a profit to  $a$  that is twice as high as  $b$ 's profit. Consequently, the AC-TAR metric favors lightweight aggregates like  $a$ . However, caching  $b$  is better for the cache performance, because the time to calculate  $b$  on the main storage is higher than the time to compute  $a$ . The AC-ETR metric also assigns a higher profit to  $a$ , but the profit is less than twice as high, since the aggregation time does not scale linearly with the number of tuples aggregated. That is why the AC-ETR metric performs up to three seconds faster than the AC-TAR metric.

Since the AC-TAR metric favors lightweight aggregates the performance of the metric decreases to the performance of the

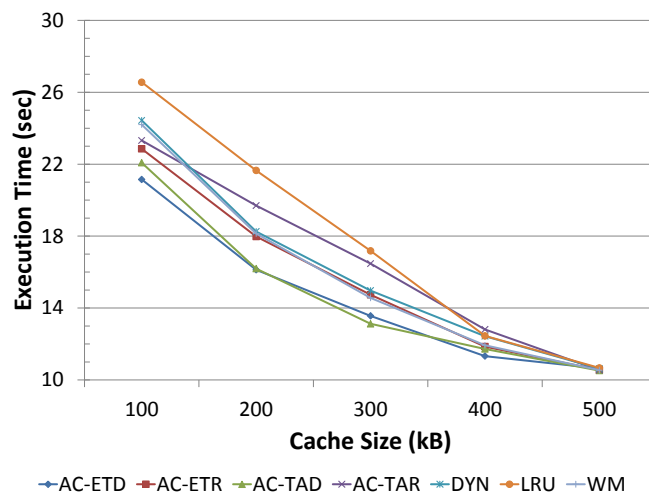


Figure 3. Comparison of Profit Metrics on Different Cache Sizes. (Delta Tuple Count 200,000, Eviction Threshold 80%)

LRU metric when the delta storage contains 300,000 tuples. The LRU metric shows the worst results because assessing an aggregate's profit on the last access only is not sufficient. The AC-TAD and AC-ETD metric constantly have results that are five to six seconds faster. That is a performance benefit of more than 35%.

### B. Cache Size

In a second experiment, we evaluate the influence of the cache size on the performance of the profit metrics. The results are shown in Figure 3. The execution time is the same for all metrics when the cache size is 500kB. Since the aggregates have a total size of 450kB, they all fit into the cache of 500kB size. Therefore, results from the runs with 500kB cache size show the optimal aggregate cache performance because no aggregate is evicted in any of the runs. In operational systems, the cache can grow several hundred gigabytes large because the systems process more than a hundred distinct queries. At a cache size of 400kB, the results of all metrics are similar because most of the aggregates fit into the cache. The cache metric has hardly any influence on the cache performance.

When the cache size is smaller than 400kB, the performance of all metrics decreases. However, the decrease significantly differs between the metrics. The LRU metric shows a performance decrease of 11 seconds or 100% in case the cache size decreases from 500kB to 200kB. In comparison, the performance decrease of the AC-ETD and the AC-TAD metrics is only five seconds or 45%. That is less than half the performance decrease. The WATCHMAN, DynaMat, and AC-ETR metric have a decrease of seven seconds or 63%. The AC-TAR metric has a decrease of 81%.

### C. Eviction Threshold

The results in Figure 4 show the impact of the eviction threshold on the profit metrics. When the eviction threshold is 0%, all aggregates are evicted from the cache once the maximum cache size of 200kB is reached. Then, the profit metrics have no influence on the cache performance so that all of them show the same performance. In general, the bigger the threshold gets, the better all of the profit metrics perform, except from the AC-TAR metric.

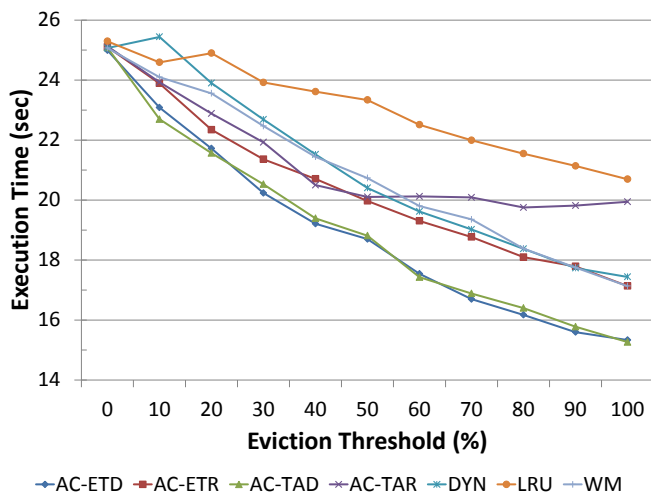


Figure 4. Comparison of Profit Metrics for Different Eviction Thresholds. (Delta Tuple Count 200,000, Cache Size 200kB)

The AC-TAR metric does not improve the cache performance, when the threshold gets bigger than 50%. That has two reasons. On the one hand, the metric tends to favor lightweight aggregates. On the other hand, costly aggregates can run multiple times between two consecutive runs of the cache trimming procedure. The bigger the intervals, the more likely is it, that the queries reoccur. Their aggregates get cached on the first occurrence. On the following occurrences, the queries are answered from the cache.

The AC-ETD and the AC-TAD metrics show the best results for an eviction threshold between 10% and 100%. They are up to two seconds faster than any of the other metrics. The AC-ETR, WATCHMAN, and DynaMat metrics show similar results when the threshold is bigger than 50%. In case the threshold is smaller, the AC-ETR metric is up to one second faster than the WATCHMAN and DynaMat metrics. The LRU metric shows the lowest performance. The time difference to the AC-ETD and the AC-TAD metrics grows from at least one second at an eviction threshold of 10% to more than five seconds at an eviction threshold of 100%.

D. Invalidation Compensation

The fourth experiment describes the impact of the invalidation compensation for deleted tuples in the main storage. The results are presented in Figure 5. We cache all queries, before we invalidate one million tuples in the main storage. After the invalidation, we reduce the cache size to 200kB and execute the analytical workload with the novel aggregate cache profit metrics. Once the metrics have the invalidation compensation factor  $icomp(q)$ ; once they do not have it. The results in Figure 5 show that the system’s performance increases by up to almost 17% when the metrics with the invalidation compensation factor are applied. The percentage increase is higher for the AC-TAR metric and the AC-ETR metric compared to the AC-ETD metric and the AC-TAD metric because the overall performance of the AC-TAR and the AC-ETR metrics is not as good as the one of the AC-ETD and AC-TAD metrics.

E. Cache Revalidation During Merge Process

In the last experiment, we analyze the performance impact of the incremental aggregate revalidation during the merge

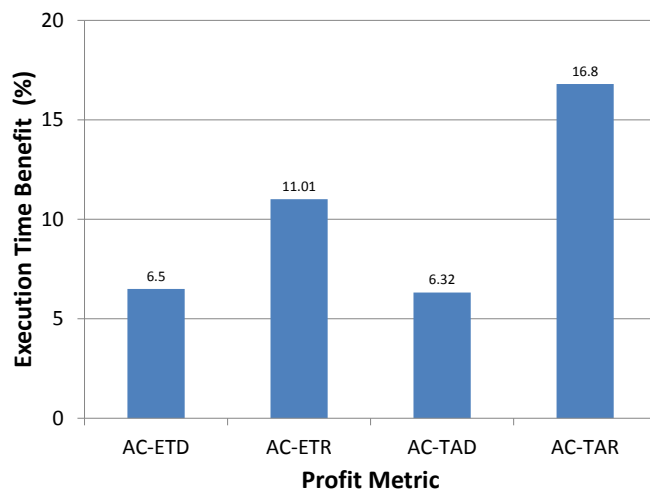


Figure 5. Impact of the Invalidation Compensation on the Profit Metrics. (Delta Tuple Count 200,000, Cache Size 200kB, Eviction Threshold 80%)

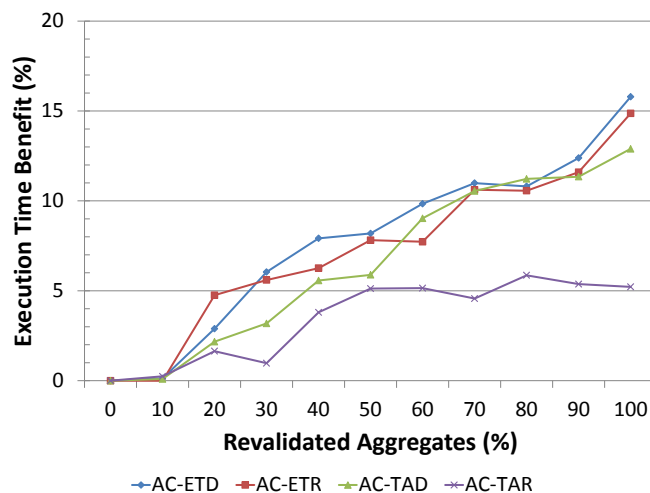


Figure 6. Impact of the Incremental Aggregate Revalidation during the Merge Process dependent from the Profit Metrics. (Delta Tuple Count 200,000, Cache Size 200kB, Eviction Threshold 80%)

process. In an operational system, where more than 100 distinct queries are executed, the revalidation of all affected aggregates can take more time than the merge process. Since, in our scenario, the merge process takes more time than the revalidation of the 100 queries, we manually limit the revalidation of the aggregates to a certain percentage of all cached aggregates. When half of the workload is processed, we trigger the merge process.

The results in Figure 6 show that the incremental revalidation yields a performance benefit of up to 15% when all aggregates are revalidated. The benefit increases with the percentage of incrementally revalidated aggregates. When the AC-TAR metric is applied, the benefit stagnates at around 5%, once more than 50% of the cached aggregates are revalidated. That has the following reason: In case no aggregates are revalidated, the cache is empty after the merge. Then, many expensive aggregates are cached and repeatedly accessed before the first cache trimming after the merge is executed. That reduces

the execution time. In case all aggregates are revalidated, the lightweight aggregates that the AC-TAR metric assigns a high profit permanently content the cache.

### VIII. CONCLUSION

We have introduced novel profit metrics and cache management algorithms tailored to the special main-delta architecture of modern IMDBs. In the evaluation section, we showed that the novel profit metrics yield up to 10% better results than existing metrics. The experiments on SanssouciDB indicate that especially the AC-ETD and the AC-TAD metrics outperform all existing profit metrics.

We also showed that it is important to consider the invalidated tuples in the main storage for the profit metrics. In our experiments, the metrics with invalidation compensation factor perform 6% to 16% better than the same metrics without invalidation compensation factor.

The evaluation also indicates that the system's performance increase when the profitable aggregates are incrementally revalidated instead of invalidated during the merge process. The performance increases up to 15% when all cached aggregates are incrementally updated.

So far our evaluation was based on a single table. In the future, we want extend the evaluation to scenarios with multiple tables and even more complex queries including joins. We also want to evaluate the impact of different incremental update strategies with our cache. The cache management system can update the cached aggregates not only during merge phase, but whenever the aggregate is touched and an on-the-fly aggregation on the delta is performed.

### REFERENCES

- [1] H. Plattner, "A Common Database Approach for OLTP and OLAP Using an In-Memory Column Database," in Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD), 2009, pp. 1–2.
- [2] —, "SanssouciDB: An In-Memory Database for Processing Enterprise Workloads Architecture of SanssouciDB," in Datenbanksysteme für Business, Technologie und Web (BTW), 2011, pp. 2–21.
- [3] S. Müller and H. Plattner, "Aggregates Caching in Columnar In-Memory Databases," in Proceedings of the International Workshop on In-Memory Data Management and Analytics (IMDM), 2013, pp. 58–69.
- [4] S. Müller, L. Butzmann, K. Howelmeyer, S. Klauck, and H. Plattner, "Efficient View Maintenance for Enterprise Applications in Columnar In-Memory Databases," in Proceedings of the IEEE International Enterprise Distributed Object Computing Conference (EDOC), 2013, pp. 249–258.
- [5] F. Hübner, J.-H. Böse, J. Krüger, C. Tosun, A. Zeier, and H. Plattner, "A cost-aware strategy for merging differential stores in column-oriented in-memory DBMS," in Proceedings of the Workshop on Business Intelligence for the Real Time Enterprise (BIRTE), 2011, pp. 38–52.
- [6] P. Scheuermann, J. Shim, and R. Vingralek, "WATCHMAN: A Data Warehouse Manager Intelligent Cache," in Proceedings of the International Conference on Very Large Data Bases (VLDB), 1996, pp. 51–62.
- [7] Y. Kotidis and N. Roussopoulos, "DynaMat: A Dynamic View Management System for Data Warehouses," ACM SIGMOD Record, vol. 28, no. 2, 1999, pp. 371–382.
- [8] C.-S. Park, M. H. Kim, and Y.-J. Lee, "Usability-based caching of query results in OLAP systems," Journal of Systems and Software, vol. 68, no. 2, 2003, pp. 103–119.
- [9] R. Cole, F. Funke, L. Giakoumakis, W. Guy, A. Kemper, K.-U. Sattler, and F. Waas, "The Mixed Workload CH-benCHmark," in Proceedings of the International Workshop on Testing Database Systems (DBTest), 2011, pp. 8:1–8:6.
- [10] S. Maffei, "Cache management algorithms for flexible filesystems," ACM SIGMETRICS Performance Evaluation Review, vol. 21, no. 2255, 1993, pp. 16–25.
- [11] H.-T. Chou and D. J. DeWitt, "An evaluation of buffer management strategies for relational database systems," in Proceedings of the International Conference on Very Large Data Bases (VLDB), 1985, pp. 127–141.
- [12] E. J. O'neil, P. E. O'neil, and G. Weikum, "The LRU-K Page Replacement Algorithm for Database Disk Buffering," ACM SIGMOD Record, vol. 22, no. 2, 1993, pp. 297–306.
- [13] T. Johnson and D. Shasha, "2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm," in Proceedings of the International Conference on Very Large Databases (VLDB), 1994, pp. 439–450.
- [14] Y. Zhou and J. F. Philbin, "The Multi-Queue Replacement Algorithm for Second Level Buffer Caches," in Proceedings of the USENIX Annual Technical Conference (USENIX ATC), 2001, pp. 91–104.
- [15] D. Lee, J. Choi, J.-H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim, "LRFU ( Least Recently / Frequently Used ) Replacement Policy : A Spectrum of Block Replacement Policies," IEEE Transactions on Computers, vol. 50, no. Technical Report, 2001, pp. 1352–1361.
- [16] S. Dar, M. J. Franklin, B. Jonsson, D. Srivastava, and M. Tan, "Semantic data caching and replacement," in Proceedings of the International Conference on Very Large Data Bases (VLDB), 1996, pp. 330–341.
- [17] J. Shim, P. Scheuermann, and R. Vingralek, "Dynamic Caching of Query Results for Decision Support Systems," in Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM), 1999, pp. 254–263.
- [18] J. Goldstein and P.-A. k. Larson, "Optimizing queries using materialized views: a practical, scalable solution," ACM SIGMOD Record, vol. 30, no. 2, 2001, pp. 331–342.
- [19] H. Plattner, A Course in In-Memory Data Management: The Inner Mechanics of In-Memory Databases. Springer, 2013.
- [20] K. Mehlhorn and P. Sanders, Algorithms and Data Structures: The Basic Toolbox. Springer, 2008.
- [21] H. Plattner and A. Zeier, In-Memory Data Management: An Inflection Point for Enterprise Applications. Springer, 2011.
- [22] F. Färber, S. K. Cha, J. Primsch, C. Bornhövd, S. Sigg, and W. Lehner, "SAP HANA database: data management for modern business applications," ACM SIGMOD Record, vol. 40, no. 4, 2011, pp. 45–51.
- [23] M. Grund, J. Krüger, H. Plattner, A. Zeier, P. Cudre-Mauroux, and S. Madden, "HYRISE: a main memory hybrid storage engine," Proceedings of the VLDB Endowment, vol. 4, no. 2, 2010, pp. 105–116.

## Survey of RDF Storage Managers

Kiyoshi Nitta  
 Yahoo JAPAN Research  
 Tokyo, Japan  
 knitta@yahoo-corp.jp

Iztok Sarnik  
 University of Primorska &  
 Institute Jozef Stefan, Slovenia  
 iztok.sarnik@upr.si

**Abstract**—This paper surveys RDF storage manager implementations that belong to a kind of database system treating locally stored RDF triples. They are systematically classified in accordance with the properties of single and multiple process technologies that include the following types: triple table, index structure, query, string translation method, join optimization method, cache, data distribution method, query process distribution method, stream process, and resource sharing architecture. This classification is applied to *3store*, *4store*, *Virtuoso*, *RDF-3X*, *Hexastore*, *Apache Jena*, *SW-Store*, *BitMat*, *AllegroGraph*, and *Hadoop/HBase*. While the classification structure is presented for each of them, detecting differences between them has become easy. The classification revealed that there will be room for further improvement of the efficient query process by developing multi-process technologies.

**Keywords**— databases; RDF databases; distributed database systems; query processing system; database system implementation.

### I. INTRODUCTION

Resource description framework (RDF) data are widely used in the Internet and their volume is growing steadily. The linked open data (LOD) project promotes the acceleration of the accumulation of RDF data to provide freely accessible on-line resources [1]. The LOD project leverages RDF's advantages by providing a data publishing principle. Each data element can be distributed to any site of the Internet. Distributed data are connected by RDF links that are also RDF data and can be located on arbitrary sites. This strategy lowers the barrier for publishers to distribute their data freely and contributes to the accumulation of a huge amount of RDF data. There are two major approaches to access those RDF data [2], [3]:

- (A-1) *Local Cache*
- (A-2) *Federated Search*

Systems based on (A-1) gather a subset of RDF data on local computational resources to accelerate the processing of queries on frequently referenced data. After accepting user queries, systems based on (A-2) distribute sub-queries to several search services distributed over the Internet and integrate replies from them to obtain updated data that is as fresh as possible. While most practical access methods may be constructed by combining these two approaches, technologies used in (A-1) will play an important role for query process efficiency. This paper surveys the challenges and solutions for developing RDF storage managers based on (A-1).

There are three notable survey papers related to RDF storage managers. The RW'11 tutorial [2] gives the most comprehensive survey about scalable RDF processing technologies

including centralized RDF repositories, distributed query processing, and scalable reasoning. This tutorial precisely explains distributed query processing system architectures that include semantic web search engines, federated systems, and P2P systems. The SIGMOD'12 tutorial [3] classified approaches for query processing over linked data into a centralized storage approach and distributed storage approach [2]. The centralized storage approach contains triple-stores based on relational database management systems, matrix, XML, and graph. Sakr and Al-Naymat [4] classified triple-stores based on relational database management systems into three categories: a) vertical (triple) table stores, b) property (n-ary) table stores, and c) horizontal (binary) table stores. This classification scheme is also explained in the above tutorials [2], [3]. The core classification structure introduced in these papers is almost the same.

Each survey paper provides a classification structure that classifies research efforts so far by focusing on the distinguishing aspects of researches. These survey papers provide useful insights and perspectives about component technologies of RDF storage managers. However, most researches implement prototype or practical systems that are equipped with combinations of useful technologies. It will be useful for researchers interested in RDF storage manager implementations to provide another type of classifications that gives several attributes to each research system. The contributions of this paper can be summarized as follows:

- Provides systematic classification of RDF storage manager implementations.
- Easily detects differences between given RDF storage manager implementations.
- Pick up attributes concerning effective processes by multi-process environments.

There are Internet pages that classify RDF storage managers. A Wikipedia page [5] provides the most comprehensive list of RDF storage managers (triplestores) with license and API function information. The W3C page [6] provides benchmark results of RDF storage managers for storing large-scale RDF data sets. While this paper provides internal functional information, those Internet pages may provide a useful perspective of RDF storage managers by combining information.

The rest of this paper is organized as follows. The Classification of RDF Storage Managers Based on Local Cache Approach section introduces a classification framework of RDF storage managers. The RDF storage managers section reports each characteristic of existing RDF storage managers using the classification framework. The Challenges section

discusses a few challenges inspired by comparing RDF storage managers. This paper is concluded in the Conclusion section.

## II. CLASSIFICATION OF RDF STORAGE MANAGERS BASED ON LOCAL CACHE APPROACH

RDF storage managers in the local cache approach can be classified in accordance with several aspects. RDF storage managers are represented by  $RSM(\mathcal{S}, \mathcal{M})$ , where  $\mathcal{S}$  and  $\mathcal{M}$  show attributes of single and multiple process issues, respectively. Attribute set  $\mathcal{S}$  has structure  $\mathcal{S}(T_s, I_s, Q_s, S_s, J_s, C_s, D_s, F_s)$ . Attribute  $T_s$  is the triple table type. It has one of the values of *vertical* ( $v$ ), *property* ( $p$ ), and *horizontal* ( $h$ ). These classes of triple tables are introduced and defined by Sakr et al. [4]. Attribute  $I_s$  is the index structure type of triple table. It has one of the values of *6-independent* ( $6$ ), *GSPO-OGPS* ( $G$ ), and *matrix* ( $m$ ). Attribute  $Q_s$  is the query type. It has one of the values of *SPARQL* ( $S$ ) and *original* ( $o$ ). Attribute  $S_s$  is the translation method type of URI and literal strings. It has one or combination of the values of *URI* ( $U$ ), *literal* ( $l$ ), *long* ( $o$ ) and *none* ( $n$ ). Values *URI* and *literal* mean ID translations of URI and literal strings, respectively. Value *long* means that only long URIs and literals are translated to identifiers. Attribute  $J_s$  is the join optimization method type. It has one of the values of *RDBMS-based* ( $R$ ), *column-store-based* ( $c$ ), *conventional-ordering* ( $o$ ), *pruning* ( $p$ ), and *none* ( $n$ ). Attribute  $C_s$  is the cache type. It has one of the values of *materialized-path-index* ( $m$ ), *reified-statement* ( $r$ ), and *none* ( $n$ ). Attribute  $D_s$  is the database engine type. It has one of the values of *RDB* ( $R$ ) and *custom* ( $c$ ). Attribute  $F_s$  is the inference feature type. It has one or combination of the values of *TBox* ( $T$ ), *ABox* ( $A$ ), and *no* ( $n$ ). While value *TBox* means inference features on the ontology level, value *ABox* means ones on the assertion level.

Attribute set  $\mathcal{M}$  has structure  $\mathcal{M}(D_m, Q_m, S_m, A_m)$ . Attribute  $D_m$  is the data distribution method type. It has one of the values of *hash* ( $h$ ), *data-source* ( $d$ ), and *none* ( $n$ ). Attribute  $Q_m$  is the query process distribution method type. It has one of the values of *data-parallel* ( $p$ ), *data-replication* ( $r$ ), and *none* ( $n$ ). Attribute  $S_m$  is the stream process type. It has one of the values of *pipeline* ( $p$ ) and *none* ( $n$ ). Attribute  $A_m$  is the resource sharing architecture type. It has one of the values of *memory* ( $m$ ), *disk* ( $d$ ), and *nothing* ( $n$ ).

Because some implementations do not disclose internal mechanisms, all attributes can have value *unknown* ( $?$ ). TABLE I shows the summary of the classification. The details are described in the next section. The values in the table correspond to the characters in parenthesis of the above description of possible values.

## III. RDF STORAGE MANAGERS

This section provides a detailed description of each RDF storage manager system. As there are many such systems, we omitted some systems due to space limitations.

### A. 3store

The attributes of this implementation are as follows:  $T_s = vertical$ ,  $Q_s = SPARQL$ ,  $S_s = string\_id$ ,  $J_s = RDBMS\_based$ ,  $D_s = RDB$ ,  $F_s = TBox$ ,  $D_m = none$ ,  $Q_m = none$ ,  $S_m = none$ . 3store [7] was originally used

for semantic web applications in particular for storing the hyphen.info RDF data set, which describes computer science research in the UK. The final version of the database consisted of 5,000 classes and about 20 million triples. 3store was implemented on top of the MySQL database management system. It included simple inferential capabilities e.g., class, sub-class, and sub-property queries, that are mainly implemented by means of MySQL queries. Hashing is used to translate URIs into an internal form of representation.

The 3store query engine used RDQL query language originally defined in the framework of the Jena project. RDQL triple expressions are first translated into relational calculus. Constraints are added to relational calculus expressions and translated into SQL. The inference is implemented by a combination of forward and backward chaining that computes the consequences of the asserted data.

### B. 4store

The attributes of this implementation are as follows:  $T_s = vertical$ ,  $Q_s = SPARQL$ ,  $S_s = string\_id$ ,  $J_s = conventional\_ordering$ ,  $D_s = RDB$ ,  $D_m = hash$ ,  $Q_m = data\_parallel$ ,  $A_m = nothing$ . 4store [8] was designed and implemented to support a range of novel applications that have emerged from the semantic web. RDF databases were constructed from web pages including people-centric information resulting from ontology with billions of RDF triples. The requirements were to store and manage  $15 \times 10^9$  triples. The design of 4store is based on 3store especially in the way RDF triples are represented.

4store is designed to operate on clusters of low-cost servers, and is implemented in ANSI C. It was estimated that the complete index for accessing quads would require around 100 GB of RAM, which was the reason for distributing data to a cluster of 64-bit multi-core x86 Linux servers with each cluster storing an RDF data partition. The cluster architecture is "shared nothing" architecture. Cluster nodes are divided into processing and storage nodes. Data segments stored on different nodes are determined by a simple formula. The formula uses resource identifiers (RID) that are indexes of URIs, literals and blank nodes. When triples are distributed to segments, RID of the triple subject is divided by number of segments. The remaining part of this calculation determines segment number of triple. One of the benefits of such a design is parallel access to RDF triples distributed to nodes holding segments of RDF data. Furthermore, segments can be replicated to distribute the total workload to the nodes holding replicated RDF data. The communication between nodes is directed by processing the nodes via TCP/IP. There is no communication between data nodes.

URIs are represented using resource identifiers that are similarly to those of 3store that were obtained by means of hashing. Triples are represented as quads. Each quad in a particular segment is stored in three indexes. Two of them are implemented using radix tries because of  $O(k)$  time complexity. The third index is used to access graphs by using a hash table.

The 4store query engine is based on relational algebra. A Rasqal SPARQL parser is used for parsing SPARQL queries. Queries are processed by SPARQL blocks. First, the

TABLE I. PROPERTIES OF RDF STORAGE MANAGERS

	$\mathcal{S}$								$\mathcal{M}$			
	$T_s$	$I_s$	$Q_s$	$S_s$	$J_s$	$C_s$	$D_s$	$F_s$	$D_m$	$Q_m$	$S_m$	$A_m$
<i>3store</i>	v		S	U	R		R	T	n	n	n	
<i>4store</i>	v		S	U	o		R		h	p		n
<i>Virtuoso</i>	v	G	S	Ulo	R		R	TA	n	n	n	
<i>RDF-3X</i>	v	6	S	UI	o		R		n	n	n	
<i>Hexastore</i>	v	6	o	UI		n			n	n	n	
<i>Apache Jena</i>	p		S	Ulo	R	r	R		n	n	n	
<i>SW-Store</i>	h			Uo	c	m	c		n	n	n	
<i>BitMat</i>	v	m	S	UI	p		c				p	
<i>AllegroGraph</i>			S				c		h	p		m
<i>Hadoop/HBase</i>	h						c			p		m

UNION expressions are collapsed. Next, FILTER expressions are evaluated. Joins are then performed on the remaining blocks. Finally, any remaining FILTERs, ORDER BYs, and DISTINCTs are applied. The primary source of optimization is the conventional join ordering. However, they also use common subject optimization and cardinality reduction. In spite of considerable work on query optimization, 4store lacks complete query optimization as it is provided by relational query optimizers.

### C. Virtuoso

The attributes of this implementation are as follows:  $T_s = vertical$ ,  $I_s = GSPO\_OGPS$ ,  $Q_s = SPARQL$ ,  $S_s = string\_id$ ,  $J_s = RDBMS\_based$ ,  $D_s = RDB$ ,  $F_s = TBox, ABox$ ,  $D_m = none$ ,  $Q_m = none$ ,  $S_m = none$ . Virtuoso [9]–[11] is a multi-model database management system based on relational database technology. Besides the functionality of the relational database management system, it also provides RDF data management, XML data management, content management, and a Web application server.

The approach of Virtuoso is to treat triple-store as a table composed of four columns. The main idea of the approach to RDF data management is to exploit existing relational techniques and add functionality to the RDBMS to deal with features specific to RDF data. The most important aspects that were considered by Virtuoso designers are: extending SQL types with the RDF data type, dealing with unpredictable sizes of objects, providing efficient indexing and extending relational statistics to cope with the RDF store based on a single table as well as efficient storage of RDF data.

The initial solution for storing RDF triples is the use of a quad table storing attributes: subject (S), predicate (P), object (O), and graph (G). Columns S, P, and G are represented as IRI ID. Column O is represented by ID only if it is longer than 12 characters. The mapping between IRIs and local IDs is stored in a table, and the mapping between the long values of O and IDs is stored in a separate table. The quad table is represented using two covering indexes based on the GSPO and OGPS attributes. Since S is the last part of OGPS we can represent it using bitmaps. We have one bitmap for one distinctive value of OGP. Compression is used on page level, which still allow random page access.

Virtuoso includes SPARQL into SQL. SPARQL queries are translated into SQL during parsing. In this way, SPARQL has all aggregation functions. SPARQL UNION is translated directly into SQL, and SPARQL OPTIONAL is translated into left outer join. Since RDF triples are stored in one quad table, relational statistics is not useful. Virtuoso uses sampling during query translations to estimate the cost of alternative plans. Basic RDF inference on TBox is done using query rewriting. For ABox reasoning Virtuoso expands semantics of owl:same-as by transitive closure.

### D. RDF-3X

The attributes of this implementation are as follows:  $T_s = vertical$ ,  $I_s = 6\_independent$ ,  $Q_s = SPARQL$ ,  $S_s = string\_id$ ,  $J_s = conventional\_ordering$ ,  $D_s = RDB$ ,  $D_m = none$ ,  $Q_m = none$ ,  $S_m = none$ . The triple-store RDF-3X reported by Neumann and Weikum [12], [13] built six independent indexes of SPO, SOP, OSP, OPS, PSO and POS (S, P, and O represent the subject, predicate, and object of the RDF triple element, respectively.) from one large triple table. The indexes are compressed using a byte-wise method that was carefully chosen to improve query process performance. They also constructed aggregated indexes for SP, PS, SO, OS, PO, and OP. They focused on join ordering to optimize the query process. The optimization uses selectivity statistics calculated for given queries using selectivity histograms and statistics of frequently accessed paths. Although it is equipped with a table to treat long URI strings as simple IDs, it has been pointed out that its translation performance was very bad [14].

They compared the RDF-3X system with PostgreSQL and MonetDB. They tried Jena2, Yars2, and Sesame 2.0, but those systems could not finish storing benchmark data in 24 hours in their experimental environment. The benchmark data contained the Barton data set ( $5.1 \times 10^7$  triples,  $1.9 \times 10^7$  IDs, and 285 types of properties), YAGO data set ( $4.0 \times 10^7$  triples,  $3.3 \times 10^7$  IDs, and 93 types of properties), and LibraryThing data set ( $3.6 \times 10^7$  triples,  $9.3 \times 10^6$  IDs, and  $3.3 \times 10^5$  types of properties). RDF-3X exceeded other systems by large margins. The source code is available for non-commercial purposes.

### E. Hexastore

The attributes of this implementation are as follows:  $T_s = vertical$ ,  $I_s = 6\_independent$ ,  $Q_s = original(customwrapped)$ ,  $S_s = strings\_id$ ,  $J_s = unknown(itseemsnone)$ ,  $C_s = none$ ,  $D_m = none$ ,  $Q_m = none$ ,  $S_m = none$ . The Hexastore [15] approach to the RDF storage system uses triples as the basis for storing RDF data. The problems of existent triple-stores investigated are the scalability of RDF databases in a distributed environment, complete implementation of the query processor including query optimization, persistent indexes, and other topics provided by database technology.

Six indexes are defined at the top of the table with three columns, one for each combination of three columns. The index used for the implementation has three levels ordered by the particular combination of SPO attributes. Each level is sorted, giving in this way the means to use ordering for optimization during query evaluation. The proposed index provides natural representation of multi-valued properties and allows fast implementation of merge-join, intersection, and union.

### F. Apache Jena

The attributes of this implementation are as follows:  $T_s = property$ ,  $Q_s = SPARQL$ ,  $S_s = string\_id$ ,  $J_s = RDBMS\_based$ ,  $C_s = reified\_statement$ ,  $D_s = RDB$ ,  $D_m = none$ ,  $Q_m = none$ ,  $S_m = none$ . In terms of the body of knowledge grown from the database community, Jena is a database programming language environment based on RDF for Java [16], [17], [18], [19]. It provides a simple abstraction and interface for manipulation of RDF graphs represented in main memory and backed by the database engine. The persistence of RDF graphs is achieved using a SQL database through a JDBC connection. Jena supports a number of database systems such as MySQL, Postgres, Oracle, and BerkeleyDB. At the core interface for manipulation of RDF graphs, Jena includes a range of RDF parsers, query language, and I/O modules for N3, N-Triple, and XML/RDF.

RDF statements are in a database-back-end of Jena represented using three tables. The URIs of resources are converted to indexes represented in one table. Larger literals are represented in another table while small literals are stored directly in a statement table. Finally, triples are stored in statement tables using indexes for resources and larger objects. Jena uses additional optimizations for fast access to common statements of a graph and reified statements. Furthermore, graphs can be stored in different sets of tables to improve the speed of query processing.

In Jena, persistence is achieved through persistent logical graphs composed of specialized graphs optimized for storing particular types of graphs. The database driver realizes access to databases abstracted away from particular database systems, each of which has their particular driver.

On a database level Jena includes three types of operations: operation 'add' that inserts new triples in a database, operation 'delete' that removes RDF statements from a database, and operation 'find' that retrieves RDF statements from a database. Query language RDQL converts SPARQL statements into a

set of find patterns including variables that can be executed as joins. While Jena1 includes a query processing engine that does not include query optimization in a database system sense, Jena2 passes on queries to the database engine. Query optimization thus takes place in a database engine. Finally, Jena2 includes mechanisms for efficient retrieval of reified statements.

### G. SW-Store

The attributes of this implementation are as follows:  $T_s = horizontal$ ,  $S_s = string\_id$ ,  $J_s = column\_store\_based$ ,  $C_s = materialized\_path\_index$ ,  $D_s = custom$ ,  $D_m = none$ ,  $Q_m = none$ ,  $S_m = none$ . Abadi proposes the use of vertical partitioning for the representation of RDF databases [20]. The advantages of using column-stores for storing RDF are: efficient representation of NULL values; efficient implementation of multi-valued attributes; support for heterogeneous records; efficient merge-joins of sorted columns; and reduced number of unions in queries. To achieve fast access to selected access paths, they are materialized.

Database management system SW-Store [21] is based on vertical partitioning of RDF data. It has been shown that storage system based on columns can significantly improve some types of queries on RDF databases. Column-oriented storage system in SW-Store is improved by using column-oriented compression; optimization for fixed length tuples; optimization of merge-join code; using column oriented query optimization; and by materialized path expressions. Empirical results support the proposed use of column-oriented store in comparison to triple-store representation and property table representation of RDF database.

While above presented novel features of column-oriented data stores are important, it seems that the most important contribution of SW-Store is to show the possibility to use simple tools like sorting and map-based indexes on large-scale distributed clusters of servers. The simplicity of tools, on one hand, and the possibility of using database technologies like query optimization on column-stores, on the other, can result in very efficient query execution.

### H. BitMat

The attributes of this implementation are as follows:  $T_s = vertical$ ,  $I_s = matrix$ ,  $Q_s = SPARQL$ ,  $S_s = string\_id$ ,  $J_s = pruning$ ,  $D_s = custom$ ,  $S_m = pipeline$ . The triple-store reported by Atre et al. [14] used a three dimensional compressed matrix index named BitMat. It avoids maintaining materialized triples as much as possible. Its query processing algorithm of SPARQL joins consists of two phases. The first phase performs efficient pruning of the triples. The second phase performs variable binding matching across the triple patterns to obtain the final results. Both two steps use compressed BitMats without any join table construction.

The BitMat index triple-store performed better than RDF-3X [12] for some queries that require managing large number of triples during join processes. They classified triple-pattern join queries to three types: a) highly selective triple patterns, b) triple patterns with low-selectivity but which generate few results, and c) low-selectivity triple patterns and low-selectivity join results. The BitMat system performed well processing type b) queries.



### I. AllegroGraph

The attributes of this implementation are as follows:  $Q_s = SPARQL$ ,  $D_s = custom$ ,  $D_m = hash$ ,  $Q_m = data\_parallel$ ,  $A_m = memory$ . AllegroGraph [22] is a triple store built on an object store AllegroCache. They are proprietary products of Franz Inc. The precise architecture of AllegroGraph has not been fully disclosed. The strongest point is its capacity limit and processing speed. A clustered version of AllegroGraph stored 1 trillion triples in about 14 days [23]. In the scalability ranking of triple stores edited by W3C [6], AllegroGraph is ranked first as of August 2011. Its benchmark experiments were performed using PC servers that have huge memories (from 48GB to 1TB). AllegroGraph does not necessarily perform materialization to process queries. We could not find enough information about how the clustered version distributes data and optimizes query processes.

### J. Hadoop/HBase

The attributes of this implementation are as follows:  $T_s = horizontal$ ,  $D_s = custom$ ,  $Q_m = data\_parallel$ ,  $A_m = memory$ . Hadoop [24] offers software environment for the implementation of large-scale distributed systems processing data on clusters of servers. It was initially designed to support fast distributed processing of very large HTML graphs. Hadoop allows for the implementation of data centers comprised of up to many 1000 servers.

The basis of Hadoop includes a set of interfaces to distributed file system, general I/O operations, RPC, serialization, and persistent data structures. A distributed file system HDFS runs on large clusters of commodity machines. MapReduce model allows for efficient manipulation of sequential data files (SequenceFile) in distributed cluster of servers. *Dictionary* data structure for indexing records based on keys (MapFile) is used to support efficient implementation of operation *map*.

Sorted sequence files and map indexes provide programming environment for the implementation database operations such as selection, projection and joins. Hadoop includes data-flow programming language Pig, which can realize some forms of classical database operations. However, sorted files and map-based index (dictionary) provide limited basis for the implementation of database structures and operations. For instance, merge joins can be implemented efficiently while index-based joins and range queries can not be implemented without extending the functionality of Hadoop.

HBase [25] is column-oriented database system implemented on top of Hadoop. It was initially based on ideas of Google's Bigtable [26]: database comprises a large set of columns describing HTML files that represent rows of the table. HBase is designed for horizontal distribution of tables into regions that are managed by one server. Map-reduce techniques can be employed to process table rows. Abadi has shown in [20] that RDF can be efficiently stored by means of column-oriented stores.

## IV. CHALLENGES

While an accompany paper provides a comprehensive list of challenges on RDF storage managers, this section discusses a few challenges inspired by viewing TABLE I.

Listed RDF repositories recorded more varied values with  $S$  attributes than with  $M$  attributes in TABLE I. Most  $M$  attributes have *none* or *unknown* values. This means that researchers so far have succeeded in achieving good performances by developing single process technologies. While practical semantic web applications tend to process large-scale data sets, solutions based on data distribution parallelism have become more popular. There will be room for further improvement of efficient query processes by developing multi process technologies considering the situation.

Caching techniques have not been researched that much. Only *Apache Jena* and *SW-Store* reported confirming the efficiency of caching techniques. Those performances depend upon types of queries and the number of different queries. Technologies for automatic investigation and classification of processing queries might become important to utilize caching technologies.

Many researches have been carried out for developing efficient join algorithms with index structures. This area has a long history in the research of database management systems [27]. While the accumulated RDF data-set is rapidly growing and SPARQL queries are basically constructed from joins of triple patterns, join operations will be applied more strongly in semantic web applications. The multi-process technologies mentioned above might produce breakthroughs of efficient join operations.

Because the standard data access method for the RDF data-set is W3C's recommendation SPARQL, most RDF storage managers can accept SPARQL queries. The semantics of SPARQL is clearly described using RDF algebra [28], [29], [30]. SPARQL-based RDF storage managers rarely cause semantic mismatch due to the existence of proposed RDF algebras. These papers also reveal that some kinds of SPARQL expressions require huge computational cost. Most of these expressions are constructed by using the *OPTIONAL* operator. While this operator was introduced to make the query language convenient enough, efficient processing of such queries will be one of the most crucial challenges of RDF storage managers.

## V. CONCLUSION

This paper surveyed the RDF storage manager implementations based on the local cache approach by introducing the systematic classification structure  $RSM(S, M)$ . This classification was applied to *3store*, *4store*, *Virtuoso*, *RDF-3X*, *Hexastore*, *Apache Jena*, *SW-Store*, *BitMat*, *AllegroGraph*, and *Hadoop/HBase*. The  $RSM$  structure was presented for each of them, so detecting the differences between them became easy. By having the  $M$  part in  $RSM$  structure, the classification revealed that there will be room for further improvement of the efficient query process by developing multi process technologies.

## VI. ACKNOWLEDGEMENT

This work was supported by the Slovenian Research Agency and the ICT Programme of the EC under PlanetData (ICT-NoE-257641).

## REFERENCES

- [1] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data - the story so far," *Int. J. Semantic Web Inf. Syst.*, vol. 5, no. 3, pp. 1–22, 2009.
- [2] K. Hose, R. Schenkel, M. Theobald, and G. Weikum, "Database foundations for scalable rdf processing," in *Proceedings of the 7th international conference on Reasoning web: semantic technologies for the web of data*, ser. RW'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 202–249. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2033313.2033317>
- [3] A. Harth, K. Hose, and R. Schenkel, "Database techniques for linked data management," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '12. New York, NY, USA: ACM, 2012, pp. 597–600. [Online]. Available: <http://doi.acm.org/10.1145/2213836.2213909>
- [4] S. Sakr and G. Al-Naymat, "Relational processing of rdf queries: a survey," *SIGMOD Rec.*, vol. 38, no. 4, pp. 23–28, Jun. 2010. [Online]. Available: <http://doi.acm.org/10.1145/1815948.1815953>
- [5] "Triplestore," <http://en.wikipedia.org/wiki/Triplestore>, 2013, [retrieved Dec. 2013].
- [6] "Largetriplestores," <http://www.w3.org/wiki/LargeTripleStores>, 2011, [retrieved Dec. 2013].
- [7] S. Harris and N. Gibbins, "3store: Efficient bulk rdf storage," in *1st International Workshop on Practical and Scalable Semantic Systems (PSSS'03)*, 2003, pp. 1–15, event Dates: 2003-10-20. [Online]. Available: <http://eprints.soton.ac.uk/258231/>
- [8] S. Harris, N. Lamb, and N. Shadbolt, "4store: The design and implementation of a clustered rdf store," in *Proceedings of the The 5th International Workshop on Scalable Semantic Web Knowledge Base Systems*, 2009.
- [9] O. Erling and I. Mikhailov, "Rdf support in the virtuoso dbms," in *CSSW*, 2007, pp. 59–68.
- [10] —, "Rdf support in the virtuoso dbms," in *Networked Knowledge - Networked Media*, ser. *Studies in Computational Intelligence*, vol. 221, 2009, pp. 7–24.
- [11] *OpenLink Virtuoso Universal Server: Documentation*, OpenLink Software Documentation Team, 2009.
- [12] T. Neumann and G. Weikum, "Rdf-3x: a risc-style engine for rdf," *Proc. VLDB Endow.*, vol. 1, no. 1, pp. 647–659, Aug. 2008. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1453856.1453927>
- [13] —, "The rdf-3x engine for scalable management of rdf data," *The VLDB Journal*, vol. 19, no. 1, pp. 91–113, Feb. 2010. [Online]. Available: <http://dx.doi.org/10.1007/s00778-009-0165-y>
- [14] M. Atre, V. Chaoji, M. J. Zaki, and J. A. Hendler, "Matrix "bit" loaded: a scalable lightweight join query processor for rdf data," in *Proceedings of the 19th international conference on World wide web*, ser. WWW '10. New York, NY, USA: ACM, 2010, pp. 41–50. [Online]. Available: <http://doi.acm.org/10.1145/1772690.1772696>
- [15] C. Weiss, P. Karras, and A. Bernstein, "Hexastore: sextuple indexing for semantic web data management," *Proc. VLDB Endow.*, vol. 1, no. 1, pp. 1008–1019, Aug. 2008. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1453856.1453965>
- [16] K. Wilkinson, C. Sayers, H. Kuno, and D. Reynolds, "Efficient rdf storage and retrieval in jena2," *Enterprise Systems and Data Management Laboratory*, HP Laboratories Palo Alto, Tech. Rep. HPL-2003-266, 2003.
- [17] B. McBride, "Jena: A semantic web toolkit," *IEEE Internet Computing*, vol. 6, no. 6, pp. 55–59, Nov. 2002. [Online]. Available: <http://dx.doi.org/10.1109/MIC.2002.1067737>
- [18] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, "Jena: implementing the semantic web recommendations," in *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, ser. WWW Alt. '04. New York, NY, USA: ACM, 2004, pp. 74–83. [Online]. Available: <http://doi.acm.org/10.1145/1013367.1013381>
- [19] A. Owens, A. Seaborne, N. Gibbins, and mc schraefel, "Clustered tdb: A clustered triple store for jena," in *WWW2009*, November 2009. [Online]. Available: <http://eprints.soton.ac.uk/266974/>
- [20] D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach, "Scalable semantic web data management using vertical partitioning," in *Proceedings of the 33rd international conference on Very large data bases*, ser. VLDB '07. VLDB Endowment, 2007, pp. 411–422. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1325851.1325900>
- [21] —, "Sw-store: a vertically partitioned dbms for semantic web data management," *The VLDB Journal*, vol. 18, no. 2, pp. 385–406, Apr. 2009. [Online]. Available: <http://dx.doi.org/10.1007/s00778-008-0125-y>
- [22] *AllegroGraph 4.11 Introduction*, Franz Inc., 2013.
- [23] "Franz' s allegrograph(r) sets new record on intel(r) xeon(r) e7 platform," [http://www.franz.com/about/press\\_room/Franz-Intel\\_6-7-11.html](http://www.franz.com/about/press_room/Franz-Intel_6-7-11.html), 2011, [retrieved Dec. 2013].
- [24] T. White, *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2009.
- [25] L. George, *HBase: The Definitive Guide*. O'Reilly Media, Inc., 2011.
- [26] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: a distributed storage system for structured data," in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 15–15. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1267308.1267323>
- [27] R. Ramakrishnan and J. Gehrke, *Database Management Systems*, 3rd ed. New York, NY, USA: McGraw-Hill, Inc., 2003.
- [28] R. Angles and C. Gutierrez, "The expressive power of sparql," in *Proceedings of the 7th International Conference on The Semantic Web*, ser. ISWC '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 114–129. [Online]. Available: [http://dx.doi.org/10.1007/978-3-540-88564-1\\_8](http://dx.doi.org/10.1007/978-3-540-88564-1_8)
- [29] J. Pérez, M. Arenas, and C. Gutierrez, "Semantics and complexity of sparql," *ACM Trans. Database Syst.*, vol. 34, no. 3, pp. 16:1–16:45, Sep. 2009. [Online]. Available: <http://doi.acm.org/10.1145/1567274.1567278>
- [30] M. Schmidt, M. Meier, and G. Lausen, "Foundations of sparql query optimization," in *Proceedings of the 13th International Conference on Database Theory*, ser. ICDT '10. New York, NY, USA: ACM, 2010, pp. 4–33. [Online]. Available: <http://doi.acm.org/10.1145/1804669.1804675>

## Design of Distributed Storage Manager for Large-Scale RDF Graphs

Iztok Savnik  
University of Primorska &  
Institute Jožef Stefan, Slovenia  
iztok.savnik@upr.si

Kiyoshi Nitta  
Yahoo Japan Research  
Tokyo, Japan  
knitta@yahoo-corp.jp

**Abstract**—Storage and management of large-scale RDF repositories is a challenge that may be compared to storage and management of large-scale HTML repositories. The main difference is in the modelling power of RDF data model comparing it to HTML hyper-graph model. RDF is much closer to the database data model and requires the capabilities of database management system rather than those offered by informational retrieval query engine. Triple-based storage systems seem to provide the functionality needed for storing RDF graphs. Triples are very natural means for the representation of graphs at various levels of abstraction. To cope with growing demand for querying large-scale triple-stores including up to several Tera triples we propose the use of massively parallel system that can be dynamically configured for particular queries into a set of parallel data-flow machines. The paper presents the design of large-scale triple-store database system *big3store*.

**Keywords**—databases; RDF databases; distributed database systems; query processing system; database system implementation.

### I. INTRODUCTION

There exists a growing interest to gather, store and query data from various aspects of human knowledge including geographical data; data about various aspects of human activities (like music, literature, and sport); scientific data (from biology, chemistry, astronomy and other scientific fields); as well as data presenting the activities of governments and other important institutions.

There is consensus that data should be presented in some form of *graph data model*, where simple and natural abstractions are used to represent data as *subjects* and their *properties* described by *objects*, that is, by means of nodes and edges of a graph. Seeing this from the point of view of knowledge developed in the fields of data modeling and knowledge representation, all existing data models and languages for the representation of knowledge can be transformed, many times very naturally, to some form of a *graph*.

There exist a number of practical projects that allow for gathering and storing graph data. One of the most famous examples is Linked Open Data (LOD) project that gathered more than 32 giga triples from the areas such as media, geography, government, life sciences and others. The language employed for the representation of data is Resource Description Framework (RDF), which is a form of graph data model.

Storing and querying such huge amounts of structured data represent a problem that could be compared to the problem of

querying huge amounts of text that appeared after the advent of Internet. The differences are in the degree of structure and semantics that data formats such as RDF and Web Ontology Language (OWL) encompass comparing them to HyperText Markup Language (HTML). HTML data published on Internet represents a huge hypergraph of documents interconnected with links. Links between documents do not carry any specific semantics except representing URIs.

Differently to HTML, RDF is a *data model* where all data are represented by means of triples (subject, predicate, object). In this format, one can represent entities and their properties in a similar way as provided in object-oriented models or AI frames. One can represent objects at different levels of abstraction: RDF can serve to model ordinary data, data modeling schemata as well as meta-data.

Primary modeling principle of RDF is assignment of special meaning to properties with selected names. In this way, we can define the exact meaning of properties that are commonly used to describe documents, persons, relationships and others. *Vocabularies* are employed to standardize the meaning of properties. For example, Dublin Core [5] project defined a set of common properties of things. Next, XML-schema [24] vocabulary defines the properties that can specify types of objects. Furthermore, vocabularies of properties and things are used to define higher-level data models realized on top of RDF. Such examples include, RDF Schema [17] as well as OWL [16] that provide object-oriented data modeling facilities and constructs for the representation of logic.

#### A. Challenges in storing and querying RDF

The amount of data in the form of triples gathered worldwide is expected to grow further towards peta (i.e.,  $10^{15}$ ) triples so that existing techniques for storing and accessing data will have to be adapted. Something similar appeared after the development of Internet, where search engines had to cope with huge hypergraph of documents including many giga documents. It seems a natural choice to tend to use similar methods to deal with the problem.

The leading idea of our approach is the use of massively parallel systems where data and query processing are distributed to many data servers. The challenges and problems that will be addressed are the following.

- 1) Definition of namespace of RDF triple-store,
- 2) Automatic distribution and replication of RDF data,

- 3) Intelligent distribution of query processing,
- 4) Dynamic updates in RDF storage manager,
- 5) Multi-threaded architecture of query executor, and
- 6) Distributed cache for query executor.

The first problem deals with definition of methods for naming entities and triples of database that will allow efficient way of managing huge amounts of structured data distributed and replicated to thousands of servers as one uniform name space. Naming schema must include ways to dynamically allocate data server for the execution of query optimally with regards to distance of data server in the network and the execution load of replicas.

The second challenge is about the design of schemas for distribution and replication of data to distributed servers so that optimal computation load is achieved. While manual distribution may seem possible, the storage of peta triples with data from all areas of human interest may require automatic distribution of data, which must be placed on distributed data server in such manner that query execution will be balanced among the servers.

Large number of servers that store distributed database require intelligent distribution of query processing to achieve appropriate response time to queries. This is covered by challenge 3. The uniform distribution schemas like hashing do not take into account semantics and structural properties of data resulting in the distribution where data on particular subject is scattered to too many data servers. The distribution of data based on semantics of data may result more efficient configuration of data servers for fast execution of queries.

In light of recent proposals for architecture of super-computers presented in [8] and by using the knowledge from the area of distributed query processing, we propose the use of global distributed query optimization, which results in optimal distributed query tree and a configuration of data servers forming a fast *dataflow machine*. Similar to super-computer systems the execution is comprised of two phases: in first phase the program or query, in our case, is optimized resulting specific dataflow machine configuration, and, in the second phase, the program executes on the specific hardware configuration, or in our case, on selected configuration of data servers.

Challenge 4 deals with updates of RDF databases. While most RDF data published through Linked Data community are stable, some portions of data are dynamically updated or found. Examples of such data would represent stock data, scientific data, or data presenting the state of institutions. With the growth of RDF databases the problem of updating RDF databases will become more important. Triple-store including very large quantities of data must be designed to provide capabilities for keeping track of changes in existing datasets as well as adding new RDF datasets.

Large-scale parallel computer systems can be recently constructed using commodity hardware that includes multi-processor systems and multi-threaded CPUs. It becomes more demanding to design triple-store architecture that maximizes query execution performance by utilizing concurrency of processes or threads running on large clusters of servers equipped with multiple processors. This problem is the topic

of challenge 5. We will provide the design of big3store, which exploits process and thread parallelism, by constructing custom parallel architecture of big3store using programming constructs of Erlang.

Finally, large-scale distribution of data and query processing in big3store calls for efficient architecture of *memory hierarchy* that will exploit locality of data. The design of local cache of data servers is presented as challenge 6. The leading idea of architecture of local cache will be its tight interrelation with query processor system, which will tend to tie data servers to particular users, and for processing particular portion of data. Data gathered in a cache of data server will contain “local” data most probably needed for processing subsequent queries assigned to a given data server.

## B. Outline

The rest of the paper is organized as follows. Section II presents architecture of RDF storage manager big3store. Storage manager is distributed to an array of servers including front servers and data servers, as described in Section II-A. Distribution of RDF database is discussed in Section II-B. Functions of front servers and data servers are described in Sections II-C and II-D. Some implementation aspects of big3store are presented in Section III. In particular, we describe distributed cache in Section III-A, distributed query execution in Section III-B, distributed query optimization in Section III-C, and architecture of dynamic updates in Section III-D. Related work is presented in Section IV and concluding remarks are given in Section V.

## II. ARCHITECTURE OF RDF STORAGE MANAGER

To provide fast access to big RDF databases and to allow heavy workload storage manager has to provide facilities for flexible distribution and replication of RDF data. Storage manager has to be re-configurable to allow many servers to work together in a cluster and to allow for different configurations of clusters to be used when executing different queries.

Storage manager for big RDF databases should be based on SPARQL and on algebra of RDF graphs [20]. To provide more general and durable storage manager its design should be based on ideas of graph databases [2]. Such a design would allow adding interfaces for popular graph data models, besides RDF, to be added later.

### A. Storage manager as cluster of data servers

Possible distribution and replication is crucial for the design of storage manager to be available globally and to provide heavy workload that is to be expected if LOD data is going to be used by masses.

Heavy distribution and replication is currently possible because of the availability of inexpensive commodity hardware for servers with huge RAM (1-100GB) and relatively large disks. The same idea was used by Google while bootstrapping and remains to be the main design direction for Google data centers [9].

As further detailed in the sequel, cluster of data servers can be easily configured into very fast data-flow machine answering a particular SPARQL query. Similar idea appears

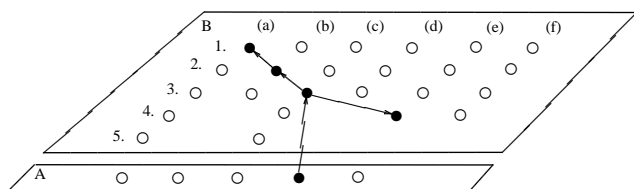


Figure 1. Configuration of servers for particular query.

recently in the area of super-computers [8], where advances in hardware technologies allow preprocessor of compiler to configure hardware facilities for a specific program. Program then runs on specially configured hardware that gains considerable speed.

The leading idea for distribution of SPARQL query processing is splitting SPARQL query into parts that are executed on different data servers in such way that the processing time of query is minimal. Data servers executing parts of SPARQL query are connected by streams of data to form cluster configuration defined for a particular SPARQL query. As with super-computers based on configuring intelligent hardware we also have strict separation between two phases: compiling the program into hardware configuration, and executing the program on selected hardware configuration.

Figure 1 presents a cluster composed of two types of servers: *front servers* represented as nodes of plane A, and *data servers* represented as nodes of plane B. Data servers are configured in *columns* labeled from (a) to (f). Complete database is distributed to columns where each column stores a portion of the complete database. The methods for the distribution of RDF data are discussed in the following sections.

Portion of database stored in a column is replicated into rows labeled from 1 to 5. The number of rows for a particular column is determined dynamically based on the query workload for each particular column. More heavy load we have on a given column more row data servers will be chosen for replication. The particular row used for execution of a query is selected dynamically based on current load of servers in a column.

A particular cluster configuration for answering a particular SPARQL query is programmed by front servers where also the optimization of SPARQL query takes place. Front server, receives SPARQL query, parses it to query tree and performs optimization based on algebraic properties of SPARQL set algebra operations. Parts of query tree are sent to internal data servers to define cluster configuration used for particular query execution.

### B. Data distribution

The schema for distribution of RDF data to a cluster of data servers has to be designed very carefully. The distribution of RDF data in local data clusters has to be transparent from outside world. Ideally, RDF data would be distributed automatically aiming to distribute transaction load optimally to data servers forming cluster on the basis of the transaction load in a given time period.

RDF data stored in a data center is distributed to *columns* of data servers that form the cluster. Each data server includes

a triple-store accessible through TCP/IP. Each column is composed of an array of data servers referred to as *rows* that are the replicas storing the same portion of big3store database.

Distribution of RDF data to columns can be defined in more ways. Firstly, data can be split manually by assigning larger datasets (databases) to columns. An example of such dataset may be dbpedia. This may be practical solution used in the initial phase of big3store implementation. Secondly, RDF data can be split to columns automatically by using SPARQL queries as the means to determine groups of RDF triples that are likely to be accessed by one query. In this context, RDFS classes are employed as the main subject of distribution as suggested in [19]. Groups of classes that are usually accessed together are assigned to *columns* where class instances are stored.

The benefits of splitting a triple store in more separate data stores (tables) has been shown by Yan et al. in [25]: queries can be executed few times faster. The reason for this can only be the size and height of indexes defined for tables representing triples. This means that few-times less blocks have to be read from database if RDF data is distributed to different tables.

There are two points where automatic reconfiguration of RDF database can be implemented. Firstly, complete database may be automatically distributed into columns as described above. Secondly, the degree of replication of portion of database stored in a column has to be determined. In other words, we have to determine how many rows (replicas) do we need to process queries targeting particular column.

### C. Front servers

Front servers are servers where SPARQL queries initiated by remote user are accepted, parsed, optimized and then distributed to data servers.

SPARQL parser checks the syntax of query and returns diagnosis to the user as well as prepare the query tree for the optimization phase. The most convenient approach to optimize SPARQL query is to transform queries into algebra and use algebraic properties for optimization. Algebra of RDF graphs [20] designed for big3store is based on work of Angles and Gutierrez [1] and the work of Schmidt et al. [22].

*Algebra of RDF graphs* reflects the nature of RDF graph data model. While it is defined on sets, the arguments of algebraic operation and its result are RDF graphs. Furthermore, expressions of RDF graph algebra are graphs themselves. Triple patterns represent the leafs of expressions. Graph patterns are expressions that stand for graphs with variables in place of some nodes and edges.

To be able to ship partial results of distributed query tree among data servers algebra of RDF graphs use operation `copy` introduced by Daniels et al. in [4]. Operation `copy` can be well integrated with operations defined on graphs due to simple set of algebraic rules that can be used for `copy`.

*Global query optimizer* will be based on rules and a form of dynamic programming algorithm for optimization of algebraic expressions [19]. Most of rules that apply to relational query optimization can be used for graph patterns. The operation `copy` also has a well defined set of rules that

can be integrated with rules for relational operations. The query optimization algorithm performs beam search guided by query cost estimation. The statistics of big3store distributed database stored with metadata server.

The result of query optimization for a given SPARQL query is a query tree where operations `copy` are placed optimally representing the points where triples are shipped from one data server to another one. The global query is therefore split into parts that are executed on different data servers. Initially, front server sends a query to a data server from a column that includes data needed to process top level of query tree. Note that all query parts are already in optimized form.

#### D. Data servers with local triple-store

In this section, we present the main features of distributed query evaluation. Firstly, we give a general view of the evaluation of distributed query. Next, we present some properties of local triple-store and the evaluation of queries in local triple-store.

##### Evaluation of distributed query

The primary job of data server is to evaluate query tree received from front server or some other data server. Query tree includes detailed information about access paths and methods for implementation of joins used for processing the query. We refer to such query tree as *annotated query tree*. Data server evaluates annotated query tree as it is without further optimization.

Triple store of data server accepts queries via TCP/IP and returns results to the return address of calling server. The communication between calling server and a given data server is realized by means of streams of triples representing results of query tree evaluation. When needed, the materialization of stream results is handled by calling server.

Query tree can include parts that have to be executed on some other data servers since data needed for a particular query part is located at some other columns. Such query parts are represented by query sub-trees with root nodes that denote operation `copy`. Again, query sub-trees can include more instances of operation `copy`, so the resulting structure of data servers constructed for a particular SPARQL query can form a tree.

Since operation `copy` is implemented by using stream of triples the query parts that form complete query tree can execute in parallel. While data server processing query sub-tree is computing the next triple to be consumed by a given data server, this data server can process previously read triple or perform some other task like accessing local triples. Moreover big3store can process many query parts in parallel functioning as a parallel dataflow machine.

##### Local evaluation of queries

Let us now present the evaluation of query on local data server. Let say that data server receives an annotated query tree `qt`. Recall that `qt` includes information about access paths to tables of triples and algorithms to be used for implementation of algebra operations.

Local triple-store includes the implementation of algebra operations and implementation of access paths, i.e., methods for accessing possibly indexed tables of triples. Algebraic operations are: selection with or without the use of index; projection; set operations union, intersection and difference; and variants of nested-loop join with or without index where the index is either index supporting equality joins or range queries.

Non-distributed storage manager for storing triples and indexes for accessing triples has to deal with very similar problems that appear in relational and object-relational storage managers. Since triple-stores are designed mainly around a table with three or four columns we propose to use existent implementation of local storage manager that implements solutions from existent relational and object-relational database technology.

We use local database management system of Erlang called *Mnesia* to store tables of triples. Mnesia includes high-level functions for accessing data stored in possibly distributed tables. Although Mnesia does not support SQL, it provides many practically useful features for distributed environment. Any table can be configured as RAM table or disk table. It supports horizontal partitioning for large tables and transaction control for distributed table operations. Tables can be reconfigured dynamically. Any Erlang object (complicated data structure) can be stored in Mnesia. If a local triple table is small enough, we might construct a fast in-memory storage using Mnesia's direct access functions. If a local table have to store large amount of triples, we might construct distributed and partitioned triple table using safe and robust transactions supporting parallel operations for distributed repositories.

Let us now present also the implementation of operation `copy` in more detail. Operation `copy` implements a stream between two data servers. The stream is realized by first initiating the execution of sub-tree of `copy` (i.e., query part) and requesting that the results are sent back to calling data server by means of a stream. On caller side access to the stream, i.e., the results of operation `copy`, is realized as access method that reads triples from the stream. that

### III. ON IMPLEMENTATION OF BIG3STORE

The initial prototype of big3store is currently under development in a high-level programming language Erlang. Erlang provides rich set of constructs convenient for distributed programming and offers abstract programming environment to allow rapid-prototyping. Successful implementation of initial prototype will allow gradual improvement of big3store efficiency that can result in a production version of the system.

#### A. Distributed cache

One of the most important principles used in database management systems is to implement some form of memory hierarchy where data read from the slow media is cached by faster media. In this way, the access to slow media may speed-up significantly. Implementation of distributed query execution on cluster of data servers with huge quantities of RAM calls for the use of memory hierarchy, i.e., exploitation of data fetched or pre-fetched from the disk and then stored in main memory.

Local cache of data servers can be in Erlang environment implemented by means of in-memory tables that store triples read from disk tables. Access to tables storing triples can be implemented by using additional database layer hiding the access to in-memory tables before reading data from disk. Such database layer would also allow seamless integration of other database management systems besides Mnesia to be plugged to query executor.

The problem of using local caches on data servers is somehow similar to the problem of scheduling on multiprocessor systems that have access to RAM via bus. After a process is executed on a particular processor the cache is loaded with data used in execution of process. Similarly, after a query tree is executed on a particular data server, cache is loaded with data used in the execution of given query.

In the case of multiprocessor scheduling, the next invocation of the same process should be executed on the same processor that includes data used in previous invocations. In the case of distributed query processing, if we select the same data server for processing the next query in the session of particular user we will most likely find some of data needed for this query already in the cache. The algorithm that selects the most appropriate row data server in a given column must therefore takes into account the affinity of user sessions to particular data servers.

The solution proposed in the area of process and thread scheduling is to use two level scheduling. On the first level process is, after creation, associated with particular processor. On the second level of scheduling the processes associated with particular processor are scheduled as in the case of uniprocessor system.

While the task of particular query can be compared to process, we can also compare the access to database system (albeit local to each data server) to the access to common RAM in the case of multiprocessor scheduling. Seeing this from the point of view of distributed query processing, user sessions are associated to data servers while we have to take care of balanced distribution of workload. This may mean that we can expand the set of data servers in a column to be employed for particular user session.

### B. Distributed query execution

Whether or not a column local repository has indexes, the whole storage management system should perform distributed query executions considering load regulations. While there are many possible solutions for the load regulation problems, Erlang/OTP programming environment may provide a convenient solution that is suitable for developing initial prototype rapidly.

In order to regulate task loads of clustered column local repositories, fixed number of `gen_server` (general server library of Erlang/OTP) processes are invoked on each physical server. A `gen_server` can update server activity codes dynamically through inter-process messages. The storage management system's bootstrap process initializes some `gen_servers` as front server processes and others as data server processes. The bootstrap process distribute data-to-data server processes according to the column configuration. When a front server process accept a query, it divides the query into

sub queries and sends them to idling data server processes according to query optimization algorithms considering efficiency and load regularity.

If a data server process should have indexes, it is a good solution to implement the process as a `gen_server` that calls Mnesia (distributed DBMS for Erlang) library functions internally for processing queries. When a data server process has to replicate to another physical server (copy operation), following steps are executed.

- 1) Find a remote physical server that has enough available CPU and memory resources (low load), and runs at least one idling data server process.
- 2) Replicate the Mnesia database instance used by the data server process to the remote physical server.
- 3) Serialize the `gen_server` implementation codes of the data server process, and install it on an idling data server process on the remote physical server.

The results of queries produced by data server processes are translated as streams. Because `gen_server` model includes message waiting loop as default functionality, it is easy to code synchronous translation of bunch of result elements. Additional codes for implementing FIFO buffer may be enough to make front and data server processes to communicate via triple streams.

### C. Distributed query optimization

Query optimization takes place on front server. SPARQL query is parsed and converted into algebra of RDF graphs. Algebra expression is converted into query tree representation, which serves as the basic data structure used in the process of optimization, cost estimation and query evaluation. The design of query processing is rooted in the design of query processor Qios [18], [19], [21].

Algebra of RDF graphs is based on relational algebra extended with operations specific for graphs. The operations are *selection*, *projection*, a form of *join*, set operations *union*, *difference* and *intersection*, and operation *optional*. Finally, algebra of triples includes operation *copy*, which allows for shipping sets of triples among data servers.

Query optimization is based on rules including pushing *projection* and *selection* towards the leafs of query tree, associativity and comutativity of *join*, rules for operations *optional* and *copy*, which integrate well with rules for relational operations. Rules are represented as patterns of query trees that stand for input and output of rule transformation.

General form of query optimization is rooted in an instance of dynamic programming technique called *memoization*. Query tree is optimized by first optimizing the children of query tree root and then by using rules to transform root of query tree. Optimized query sub-trees are inserted into the appropriate *equivalence classes* and their cost is stored for further use. Since the space of all hypotheses (query trees) is too big to explore completely, sub-optimal additions to the basic form of dynamic programming can be employed. For instance *beam search* selects at each point of optimization only the most promising alternatives for query transformation.

#### D. Architecture for managing dynamic updates

Although most amount of RDF data are stable, some data are dynamically updated. It makes difficult for data manager to build index on the set including updated data. If data managers are distributed on cluster of several PC servers, data statistics and query patterns strongly influence means for distributing and caching data over the cluster. The data might change or grow dynamically. The system load may also influence distribution configuration and cache lifetime. These make the problem more complicated. Occasionally, new RDF links might be discovered through unrelated search processes.

RDF repositories should include efficient means for accessing dynamically updated data. If elements in a data-set are updated frequently, computational cost of indexing the data-set may exceed speed-up benefit of accessing operations. We will have the threshold updating frequency by investigations with practical experiments. Because columns exceeding the threshold should not have indexes, they should be stored in a special type of repository for efficient retrieval.

One possible solution is to implement triple-store local to *column* by a set of tiny proactive on-memory processes. Such repositories can be easily coded using Erlang programming language. Adding, deleting, and modifying operations only require accesses to the target triple-store processes. The manager of column triple-store can broadcast query messages to local triple-store processes. Each process may respond asynchronously to the caller, if the query matches its contents. For the manager receives answers asynchronously, it can provide query results as a stream to its caller process. Copy operations can be easily implemented using the process dictionary serialization function of Erlang programming language. Triple-store processes execute copy or modification reflection operations independently. If the process is coded to execute those operations only under low load situations, the repository may have a method for high load tolerance.

#### IV. RELATED WORK

This section presents some of more important systems for querying RDF data including: 3store, 4store, Virtuoso, RDF-3X, and Hexastore; see survey presented in [14] for a more complete overview of RDF storage managers.

*a) 3store:* 3store [10] was originally used for semantic web applications in particular for storing hyphen.info RDF dataset describing computer science research in UK. Final version of database consisted of 5000 classes and about 20 million triples. 3store was implemented on top of MySQL database management system. It included simple inferential capabilities, e.g. class, sub-class, and sub-property queries mainly implemented by means of MySQL queries. Hashing is used to translate URIs into internal form of representation.

Query engine of 3store used RDQL query language originally defined in frame of Jena project. RDQL triple expressions are first translated into relational calculus. Constraints are added to relational calculus expressions and they are translated into SQL. Inference is implemented by a combination of forward and backward chaining computing the consequences of asserted data.

*b) 4store:* 4store [11] was designed and implemented to support a range of novel applications emerged from semantic web. RDF databases were constructed from web pages including people-centric information resulting ontology with billions of RDF triples. The requirements were to store and manage  $15 \times 10^9$  triples.

4store is designed to operate on clusters of low-cost servers. It is implemented in ANSI C. It was estimated that the complete index for accessing quads would require around 100 GB of RAM, which was the reason to distribute data to a cluster of 64-bit multicore x86 Linux servers each storing a partition of RDF data. The architecture of cluster uses "Shared Nothing" architecture. Cluster nodes are divided into processing and storage nodes. Data segments stored on different nodes are determined by a simple formula calculating RID of subject modulo number of segments. The benefits of such design is parallel access to RDF triples distributed to nodes holding segments of RDF data. Furthermore, segments can be replicated to distribute total workload to the nodes holding replicated RDF data. The communication between nodes is directed by processing nodes via TCP/IP. There is no communication between data nodes.

The 4store query engine is based on relational algebra. Primary source of optimization is the conventional ordering on the joins. However, they also use common subject optimization and cardinality reduction. In spite of considerable work on query optimization, 4store lacks complete query optimization as it is provided by relational query optimizers.

*c) Virtuoso:* Virtuoso [6], [7], [15] is a multi-model database management system based on relational database technology. The approach of Virtuoso is to treat triple-store as a table composed of four columns. The main idea of the approach to management of RDF data is to exploit existing relational techniques and to add functionality to RDBMS in order to deal with features specific to RDF data. The most important aspects that were considered by Virtuoso designers are: extending SQL types with RDF data type, dealing with unpredictable sizes of objects, providing efficient indexing and extending relational statistics to cope with RDF store based on single table, as well as efficient storage of RDF data.

Virtuoso integrates SPARQL into SQL. SPARQL queries are translated into SQL during parsing. SPARQL has in this way all aggregation functions. SPARQL union is translated directly into SQL and SPARQL optional is translated into left outer join. Since RDF triples are stored in one quad table, relational statistics is not useful. Virtuoso uses sampling during query translations to estimate the cost of alternative plans. Basic RDF inference on TBox is done using query rewriting. For ABox reasoning Virtuoso expands semantics of owl:same-as by transitive closure.

*d) RDF-3X:* Triple-store RDF-3X presented by Neumann and Weikum [12], [13] builds 6 independent indexes of SPO, SOP, OSP, OPS, PSO and POS (for subject, property and object columns) from one large triple table. The indexes are compressed using a byte-wise method that was carefully chosen to improve query process performance. Join re-ordering is used to optimize query process. The optimization uses selectivity statistics calculated for given queries using selectivity histograms and frequent path statistics. Although it equips a



table to treat long URI strings as simple ids, Atre et al. in [3] points that its search performance was very bad. RDF-3X system was compared with PostgreSQL and MonetDB. The benchmark data contained Barton data. RDF-3X exceeded other systems with large margins. The source code is available for non-commercial purposes.

e) *Hexastore*: Hexastore [23] approach to RDF storage system uses triples as the basis for storing RDF data. The problems of existent triple-stores pursued are the scalability of RDF databases in distributed environment, and complete implementation of query processor including query optimization, persistent indexes, and other topics provided by database technology.

Six indexes are defined on top of table with three columns, one for each combination of three columns. Index used for the implementation has three levels ordered by particular combination of SPO attributes. Each level is sorted giving in this way the means to use ordering for optimizations during query evaluation. Proposed index provides natural representation of multi-valued properties, and it allows fast implementation of merge-join, intersection and union.

## V. CONCLUSION

The design of large-scale storage manager for RDF is presented in the paper. The presented work is focused to the definition of most important design directions and implementation decisions of big3store. Hardware architecture of big3store is based on massive parallel array of data servers arranged into columns. Rows of columns are replicas, i.e., data servers that store a portions of big3store database. Distribution of complete big3store database is guided by semantic information used to group RDF triples.

The initial prototype in Erlang programming environment is currently under development. Erlang provides efficient programming constructs for implementation of massively parallel systems. Distributed query evaluation system of big3store, for instance, will use processes to represent query nodes that stand for operations of algebra of RDF graphs. SPARQL queries, optimized by means of programming technology provided by relational database systems, are translated to data-flow machine composed of Erlang processes. Therefore, array of distributed data servers becomes resource for optimized allocation of data-flow machines executing individual queries.

## ACKNOWLEDGMENT

This work was supported by the Slovenian Research Agency and the ICT Programme of the EC under PlanetData (ICT-NoE-257641).

## REFERENCES

- [1] R. Angles and C. Gutierrez. The expressive power of sparql. In *Proceedings of the 7th International Conference on The Semantic Web, ISWC '08*, pages 114–129, Berlin, Heidelberg, 2008. Springer-Verlag.
- [2] R. Angles and C. Gutierrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1):1:1–1:39, Feb. 2008.
- [3] M. Atre, V. Chaoji, M. J. Zaki, and J. A. Hendler. Matrix "bit" loaded: a scalable lightweight join query processor for rdf data. In *Proceedings of the 19th international conference on World wide web, WWW '10*, pages 41–50, New York, NY, USA, 2010. ACM.
- [4] D. Daniels, P. G. Selinger, L. M. Haas, B. G. Lindsay, C. Mohan, A. Walker, and P. F. Wilms. Introduction to distributed query compilation in r\*. IBM Research Report RJ3497 (41354), IBM, June 1982.
- [5] Dublin core metadata initiative. <http://dublincore.org/>, 2013.
- [6] O. Erling and I. Mikhailov. Rdf support in the virtuoso dbms. In *CSSW*, pages 59–68, 2007.
- [7] O. Erling and I. Mikhailov. Rdf support in the virtuoso dbms. In *Networked Knowledge - Networked Media*, volume 221 of *Studies in Computational Intelligence*, pages 7–24, 2009.
- [8] M. J. Flynn, O. Mencer, V. Milutinovic, G. Rakocevic, P. Stenstrom, R. Trobec, and M. Valero. Moving from petaflops to petadata. *Commun. ACM*, 56(5):39–42, May 2013.
- [9] S. Ghemawat, H. Goto, and S.-T. Leung. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles, SOSP '03*, pages 29–43, New York, NY, USA, 2003. ACM.
- [10] S. Harris and N. Gibbins. 3store: Efficient bulk rdf storage. In *1st International Workshop on Practical and Scalable Semantic Systems (PSSS'03)*, pages 1–15, 2003. Event Dates: 2003-10-20.
- [11] S. Harris, N. Lamb, and N. Shadbolt. 4store: The design and implementation of a clustered rdf store. In *Proceedings of the 5th International Workshop on Scalable Semantic Web Knowledge Base Systems*, 2009.
- [12] T. Neumann and G. Weikum. Rdf-3x: a risc-style engine for rdf. *Proc. VLDB Endow.*, 1(1):647–659, Aug. 2008.
- [13] T. Neumann and G. Weikum. The rdf-3x engine for scalable management of rdf data. *The VLDB Journal*, 19(1):91–113, Feb. 2010.
- [14] K. Nitta and I. Savnik. Survey of rdf storage managers. Technical Report (In preparation), Yahoo Japan Research & FAMNIT, University of Primorska, 2013.
- [15] OpenLink Software Documentation Team. *OpenLink Virtuoso Universal Server: Documentation*, 2009.
- [16] Owl 2 web ontology language. <http://www.w3.org/TR/owl2-overview/>, 2012.
- [17] Rdf schema. <http://www.w3.org/TR/rdf-schema/>, 2004.
- [18] I. Savnik. Qios: Querying and integration of internet data. <http://osebje.famnit.upr.si/savnik/qios/>, 2009.
- [19] I. Savnik. On using object-relational technology for querying lod repositories. In *The Fifth International Conference on Advances in Databases, Knowledge, and Data Applications, DBKDA 2013*, pages 39–44, Jan. 2013. Dates: from January 27, 2013 to February 1, 2013.
- [20] I. Savnik and K. Nitta. Algebra of rdf graphs. Technical Report (In preparation), FAMNIT, University of Primorska, 2013.
- [21] I. Savnik, Z. Tari, and T. Mohoric. Qal: A query algebra of complex objects. *Data & Knowledge Engineering*, 30(1):57 – 94, 1999.
- [22] M. Schmidt, M. Meier, and G. Lausen. Foundations of sparql query optimization. In *Proceedings of the 13th International Conference on Database Theory, ICDT '10*, pages 4–33, New York, NY, USA, 2010. ACM.
- [23] C. Weiss, P. Karras, and A. Bernstein. Hexastore: sextuple indexing for semantic web data management. *Proc. VLDB Endow.*, 1(1):1008–1019, Aug. 2008.
- [24] Xml schema. <http://www.w3.org/XML/Schema>, 2012.
- [25] Y. Yan, C. Wang, A. Zhou, W. Qian, L. Ma, and Y. Pan. Efficient indices using graph partitioning in rdf triple stores. In *Proceedings of the 2009 IEEE International Conference on Data Engineering, ICDE '09*, pages 1263–1266, Washington, DC, USA, 2009. IEEE Computer Society.

# Learning Links in MeSH Co-occurrence Network: Preliminary Results

Andrej Kastrin

Faculty of Information Studies

Novo mesto, Slovenia

Email: andrej.kastrin@guest.arnes.si

Dimitar Hristovski

Institute of Biostatistics and Medical Informatics

Faculty of Medicine, University of Ljubljana

Ljubljana, Slovenia

Email: dimitar.hristovski@gmail.com

**Abstract**—Literature-based discovery (LBD) is focusing on automatically generating scientific hypotheses by uncovering hidden, previously unknown relations between existing knowledge. Co-occurrences between biomedical concepts can be represented by a network that consists of a set of nodes representing concepts and a set of edges representing their relationships. In this work we propose a method for link prediction of implicit connections between Medical Subject Headings (MeSH<sup>®</sup>) descriptors. Our approach is complementary to standard LBD. Link prediction was performed using Jaccard and Adamic-Adar similarity measures. Preliminary results showed high prediction performance with area under the ROC curve of 0.78 and 0.82 for Jaccard and Adamic-Adar coefficient, respectively.

**Index Terms**—network analysis, link prediction, literature-based discovery

## I. INTRODUCTION

Retrieval and linking different chunks of scientific information into understandable and interpretable knowledge becomes a challenging task. Text mining technologies complement manual information retrieval from biomedical sources [1]. Common text mining tasks in biomedicine include the recognition of explicit facts from the literature, document summarization, question answering and literature-based discovery (LBD).

LBD is a methodology for automatically generating hypotheses for scientific research by uncovering hidden, previously unknown relationships between existing knowledge [2]. The LBD methodology was pioneered by Swanson [3], who proposed that dietary fish oils might be used to treat Raynauds disease because they lower blood viscosity, reduce platelet aggregation and inhibit vascular reactivity. The basic assumption of Swansons approach is that there exists two scientific domains that do not communicate. A segment of knowledge in one domain may be related to knowledge in the other domain, but this relationship is unknown. The methodology of LBD relies on the idea of concepts relevant to three literature domains: X, Y, and Z. For example, suppose a researcher has found relationship between disease X and a gene Y. Further suppose that a separate researcher has studied the effects of substance Z on gene Y. The use of LBD may suggest an XZ relationship, indicating that substance Z may potentially treat disease X.

Associations between literature entities based on co-occurrence of biomedical terms, such as diseases or genes constitute an important part of knowledge representation. A co-

occurrence approach is built on the assumption that biomedical concepts occurring together in the same title or abstract are in some way biologically related [4], [5]. Biomedical knowledge can be thus viewed as a set of concepts along with the relations between them. Interactions between concepts can be described in terms of a graph, consisting of nodes and edges, where the former represent concepts and the latter represent their relationships. Knowledge network is not static. It is a dynamic structure that evolves over time either by addition of new nodes or by new links that form between nodes.

Link prediction is a newly emerging research field that is at the intersection of the network analysis and machine learning. Understanding the mechanisms of link formation in complex networks is a long standing challenge for network analysis. Link prediction refers to the discovery of future links between nodes that are not directly connected in the current snapshot of a given network [6]. Seen in this way, the link prediction problem is similar to LBD. In the literature several link prediction techniques have been proposed. These techniques can be used to predict new link formation by estimating the likelihood of link formation between two nodes on the basis of the observed network topology.

In this work we propose a method for link prediction in biomedical domain, i.e. for prediction and evaluation of implicit or previously unknown connections between biomedical concepts. Our approach is complementary to the traditional LBD. To evaluate the link prediction techniques for LBD, here we investigate the performance of link prediction techniques for networks obtained from Medical Subject Headings (MeSH<sup>®</sup>) [7] co-occurrence data.

## II. METHODS

### A. Basic Terminology

A network is represented by a graph  $G(V, E)$  that consists of a set of nodes  $V$  representing concepts and a set of edges  $E$  representing relationships between the nodes [8]. The number of edges of a node  $i$  is denoted by its degree  $k_i$ .

The link prediction problem can be formally represented as follows. Suppose we have a network  $G[t_1, t_2]$  which contains all interactions among nodes that take place in the time interval  $[t_1, t_2]$ . Further suppose that  $[t_3, t_4]$  is a time interval occurring after  $[t_1, t_2]$ . The task of link prediction is to provide a list of edges that are present in  $G[t_3, t_4]$  but absent in  $G[t_1, t_2]$ . We

refer to  $G[t_1, t_2]$  as the train network and  $G[t_3, t_4]$  as the test network (Figure 1). In this work the prediction was performed on a core subnetwork which consists of nodes which have at least  $k = 3$  neighbors.

### B. Data Collection and Network Construction

The train and test networks were constructed from the Unified Medical Language System (UMLS<sup>®</sup>) [9] co-occurrence table (MRCOC). MRCOC table includes statistical aggregations of co-occurrences of biomedical concepts in different data sources. Two overall frequencies of MEDLINE<sup>®</sup> co-occurrence are provided: one for recent MEDLINE data (MED) and one for MEDLINE data from a preceding block of years (MBD). MRCOC provides the frequency of co-occurrence of two concepts in the same indexed articles from MEDLINE (i.e., the number of articles discussing both concepts during a given time period). We select only those co-occurrence pairs that refer to MeSH descriptors.

The constructed networks were post-processed to remove all non-useful edges. We applied the Pearsons chi-square ( $\chi^2$ ) test for independence for each co-occurrence pair to obtain a statistic, which indicates whether a particular pair of MeSH descriptors occurs together more often than by chance [10]. If  $\chi^2$  is greater than the critical value of 3.84 ( $p \leq 0.05$ ), we can be 95% confident that a particular MeSH relation occurs more often than by chance.

### C. Experimental Setup

The link prediction framework we use follows the procedure first introduced by Liben-Nowell and Kleinberg [11]. We perform link prediction using proximity measures. Proximity measures are used to find similarity between a pair of nodes. For each node pair  $(u, v)$ , a link prediction method gives score  $s(u, v)$ , an estimate of the likelihood of link formation between nodes  $u$  and  $v$ . Among various proximity measures proposed in the literature we use Jaccard and Adamic-Adar coefficients. Jaccard coefficient measures the probability that a neighbor of  $u$  or  $v$  is a neighbor of both  $u$  and  $v$  [12]. Jaccard coefficient simply divides the number of common neighbors by the number of total neighbors. Adamic-Adar coefficient measures neighborhood overlap between nodes  $u$  and  $v$ , weighting the overlap of smaller neighborhoods more heavily [13].

We examine how accurately we can predict which node pairs will connect between times  $t_3$  and  $t_4$  despite not having any co-occurrences before time  $t_3$ . The major challenge in prediction evaluation is the huge number of possible node pairs which can greatly increase computational time. To cope with this issue we use bootstrap resampling approach [14]. We use 100 bootstrap samples. In each bootstrap step we draw a random sample of 1000 nodes and create appropriate train and test subgraphs. Next, we compute the link prediction score  $s(u, v)$  for each node pair  $(u, v)$  that is not associated with any interaction before time  $t_3$  by using one of the link prediction techniques introduced in the previous paragraph. We assign class label ‘positive’ to this node pair if the

TABLE I  
BASIC TOPOLOGICAL CHARACTERISTICS OF THE MESH NETWORKS.

Parameter	MBD network	MED network
Density	0.01	0.01
Mean degree	274.78	298.05
Average path length	2.23	2.20
Clustering coefficient	0.27	0.26
Small-worldness index	21.57	20.70

connection occurs in test network and ‘negative otherwise’. The prediction performance was evaluated using the receiver operating characteristic (ROC) curves. A ROC graph depicts relative tradeoffs between true positives and false positives. As a measure of prediction performance we use the area under the ROC curve (*AUC*). The *AUC* is a widely used performance measure which can be interpreted as the probability that a randomly selected link is given a higher link prediction score than a randomly selected non-existent link. Final *AUC* value was averaged over 100 bootstrap samples.

### III. RESULTS

Before evaluating the effectiveness of link prediction techniques, we describe the characteristics of the used datasets. The MBD network consists of 24,225 nodes and 4,897,380 edges. We filter out all non-useful edges using  $\chi^2$  test. After reduction the MBD network contains 3,328,288 edges. The MED network contains 25,570 nodes and 5,615,965 edges. After the filtering step, the number of edges decreased to 3,810,535. The topological properties of both networks are summarized in Table I. Both networks are very similar regarding topological properties. The networks exhibit relatively short average path length between all pairs of nodes. On average there are only about two hops from the selected node to any other node. Both networks exhibit small world property because of small average path length and relatively high clustering.

The classification performance is summarized in Figure 2. Mean *AUC* for Jaccard coefficient was  $AUC = 0.78$  with  $SD = 0.02$ . Mean *AUC* for Adamic-Adar coefficient was  $AUC = 0.82$  with  $SD = 0.01$ . Adamic-Adar coefficient exhibited slightly better performance than Jaccard coefficient.

### IV. CONCLUSION AND FUTURE WORK

In this paper, we apply and evaluate link prediction methods on a network based on co-occurrence patterns between MeSH descriptors. We have exploited two methods for link prediction task, namely Jaccard and Adamic-Adar coefficients and demonstrated that link prediction is plausible with high prediction performance. To the best of our knowledge, this is the first work that investigates unsupervised learning for link prediction of literature-derived network in the biomedical domain.

There are many possible directions for future work. One is to consider addition similarity measures, including preferential attachment, Katz measure, SimRank, or cosine similarity, to

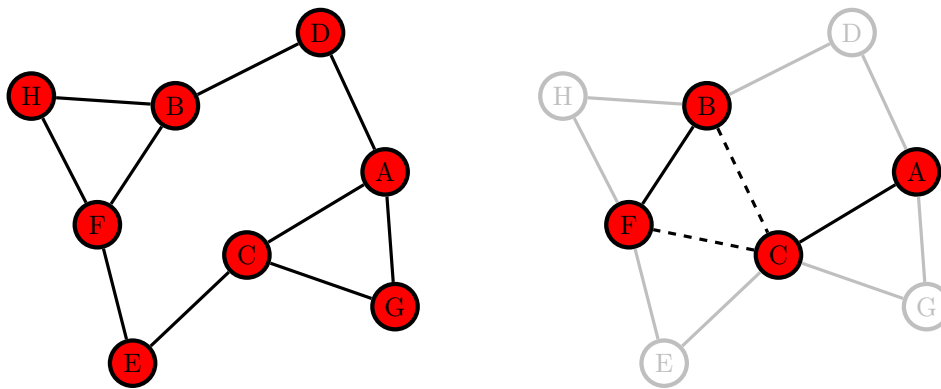


Fig. 1. Train (left) and test (right) network. New link formation is predicted from the topology of a network obtained from co-occurrences in the training period. We investigate the performance of link prediction techniques by comparing the predicted links with actual new links within the testing period. Prediction and evaluation was performed on a core subnetwork which consists of nodes which have at least  $k = 3$  neighbors.

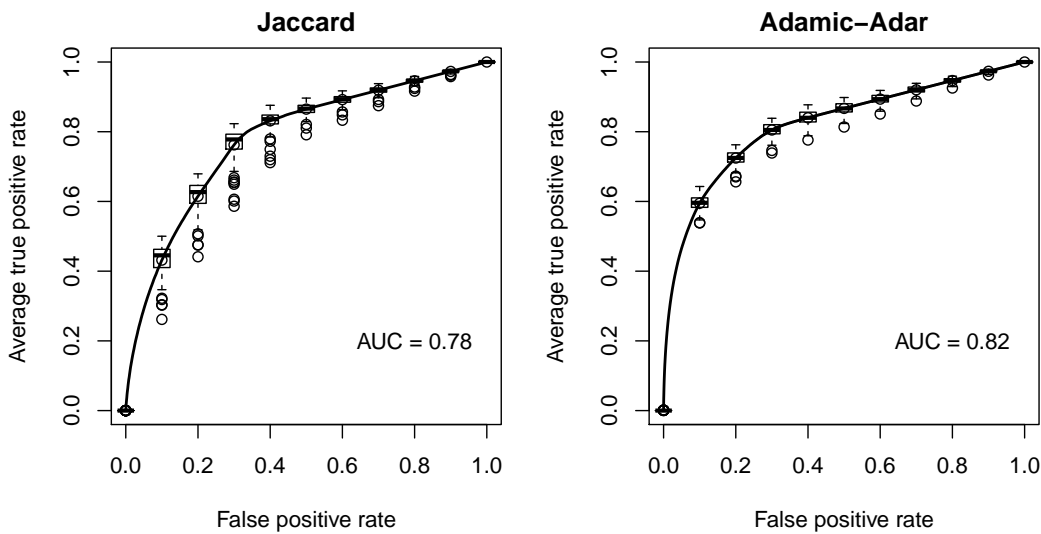


Fig. 2. ROC curves.

name just a few. Second, we should investigate prediction performance of link prediction algorithm on different networks. Biomedical science is full of interesting complex networks; for example we should consider NCBI gene or protein network, KEGG collection, or UniProt database.

Further we should investigate the connection between network attributes and prediction performance. It is well known that nodes which share similar properties tend to create links to each other. For example, persons in a social network who share similar interests are very likely to be friends [15]. In our case we could use attributes such as number of occurrences of particular MeSH descriptor, semantic type from UMLS Semantic Network, etc. We should also systematically analyze the influence of network topology on prediction performance. The process of link creation may be a result of the joint influence of several mechanisms such as small world effect and preferential attachment.

ACKNOWLEDGMENT

This work was supported by Slovenian Research Agency.

REFERENCES

- [1] D. Rebolzh-Schuhmann, A. Oelrich, and R. Hoehndorf, "Text-mining solutions for biomedical research: Enabling integrative biology," *Nature Reviews. Genetics*, vol. 13, no. 12, pp. 829–839, 2012.
- [2] D. Hristovski, T. Rindflesch, and B. Peterlin, "Using literature-based discovery to identify novel therapeutic approaches," *Cardiovascular & Hematological Agents in Medicinal Chemistry*, vol. 11, no. 1, pp. 14–24, 2013.
- [3] D. R. Swanson, "Fish oil, Raynaud's syndrome, and undiscovered public knowledge," *Perspectives in Biology and Medicine*, vol. 30, no. 1, pp. 7–18, 1986.
- [4] B. Stapley and G. Benoit, "Biobibliometrics: Information retrieval and visualization from co-occurrences of gene names in Medline abstracts," *Pacific Symposium on Biocomputing*, vol. 5, pp. 526–537, 2000.
- [5] R. Frijters, M. van Vugt, R. Smeets, R. van Schaik, J. de Vlieg, and W. Alkema, "Literature mining for the discovery of hidden connections between drugs, genes and diseases," *PLoS Computational Biology*, vol. 6, no. 9, p. e1000943, 2010.
- [6] L. Lü and T. Zhou, "Link prediction in complex networks: A survey," *Physica A: Statistical Mechanics and its Applications*, vol. 390, no. 6, pp. 1150–1170, 2011.

- [7] M. H. Coletti and H. L. Bleich, "Medical subject headings used to search the biomedical literature." *Journal of the American Medical Informatics Association : JAMIA*, vol. 8, no. 4, pp. 317–323, 2001.
- [8] M. E. J. Newman, "The structure and function of complex networks," *SIAM Review*, vol. 45, no. 2, pp. 167–256, 2003.
- [9] D. A. Lindberg, B. L. Humphreys, and A. T. McCray, "The Unified Medical Language System." *Methods of information in medicine*, vol. 32, no. 4, pp. 281–91, 1993.
- [10] C. D. Manning and H. Schuetze, *Foundations of statistical natural language processing*. Cambridge, MA: MIT Press, 1999.
- [11] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *Journal of the American Society for Information Science and Technology*, vol. 58, no. 7, pp. 1019–1031, 2007.
- [12] A. Rajaraman and J. D. Ullman, *Mining of massive datasets*. New York, NY: Cambridge University Press, 2011.
- [13] L. A. Adamic and E. Adar, "Friends and neighbors on the Web," *Social Networks*, vol. 25, no. 3, pp. 211–230, Jul. 2003.
- [14] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: Data mining, inference, and prediction*. New York, NY: Springer, 2011.
- [15] Z. Liu, J.-L. He, K. Kapoor, and J. Srivastava, "Correlations between community structure and link formation in complex networks," *PLoS ONE*, vol. 8, no. 9, p. e72908, 2013.