Resource Aware Workload Management for Autonomic Database Management Systems

Wendy Powley, Patrick Martin, Natalie Gruska
School of Computing
Queen's University
Kingston, Ontario, Canada
{wendy, martin, gruska}@cs.queensu.ca

Paul Bird, David Kalmuk
IBM
Markham Lab
Markham, Ontario, Canada
{pbird, dckalmuk}@ca.ibm.com

Abstract **-- Workloads running in a multi-purpose database environment often compete for system resources causing resource contention, which leads to poor performance. Autonomic database systems will be required to recognize that the system resources are not being utilized optimally and take action to correct the situation. Workload management techniques can be used to choose an appropriate mix of concurrent work to reduce resource contention. We describe a resource aware scheduling approach that predicts the amount of CPU, I/O and sort heap memory that will be required by a query and schedules each query to run only when doing so is unlikely to overwhelm the resources. We present experimental evidence that indicates that overall system performance can be improved using this technique.**

Keywords- workload management; database management systems; autonomic computing; scheduling

I. INTRODUCTION

Database management systems (DBMSs) are an integral part of virtually every computing system and with modern day demands on such systems to handle diverse data types, mixed workloads, and ever-changing demand, it is more important than ever to ensure that these complex systems are self-managing and self-optimizing. It is no longer feasible to manually reconfigure a system to handle a new workload type or a change in workload intensity. The system itself must recognize changing conditions and adapt accordingly. Maintaining a balance of work in the system is crucial to ensure that all the demands and goals are met.

The characteristics of a database workload determine how the resources are used. Online Analytical Processing (OLAP) workloads, for instance, may access a large quantity of data, perform complex calculations and sort large quantities of data thus taxing the CPU, the I/O subsystem and the sort memory. Transactional workloads, on the other hand, may simply scan a table for a particular result and use very little CPU or sort memory. Two or more workloads with similar characteristics running concurrently on the DBMS can result in workload interference, often due to resource contention.

Workload interference may lead to performance degradation in the DBMS system. Consider a workload that is currently executing 300 transactions per second and is using 98% of the CPU. If another workload begins executing on the system that is also CPU-heavy, the CPU will become overloaded. The work will continue to be processed, but at a slower speed as the CPU must be shared. The performance of the initial workload will degrade, perhaps violating goals that have been defined for this workload. If the competing workload was sort intensive and CPU-light, the two workloads may have executed in harmony without detrimental effects to the initial workload.

Workload control is the process whereby the DBMS exerts control over which work is allowed to run in the system. This may be done by admission control (deciding whether or not a query will be admitted to the system based on some criteria), scheduling (deciding the order that the admitted queued queries will be allowed to run) or execution control (termination, suspension, or throttling of currently executing queries) [9]. An autonomic database system incorporates workload control to ensure that the system runs in an optimal state where resources are used effectively and efficiently while allowing all work to meet its service level objectives.

Over the past several years, we have developed a number of workload management techniques [5] [6] [7] [8] and defined a framework that combines the various techniques into a unified system for autonomic workload management [10]. The work described in this paper is a subset of our framework and involves a scheduling approach to workload management. In previous work [4], we proposed a method of scheduling queries based on estimates of the amount of sort heap memory required by each query. The present work extends this work to add additional resources, namely CPU and I/O, and bases the scheduling decisions on the predicted usage of all three resources. The goal of our work is to schedule database queries such that the order of execution ensures that system resources are utilized as fully as possible while not overloading any one resource.

The remainder of the paper is structured as follows. Section II outlines related work. Section III describes the architecture for our prototype scheduling system and outlines the approach. Section IV presents experimental validation of the work. In Section V, we present the conclusions and future directions.

II. RELATED WORK

The current work focuses on scheduling as a form of workload control for database systems. Many algorithms

 ISBN: 978-1-61208-331-5

such as first-come-first-served, shortest job first and priority scheduling are used in operating system job scheduling [16]. We make use of the first fit algorithm in our scheduling of database queries in the current research.

Modeling approaches to predict performance metrics for database queries are gaining in popularity [15]. These performance metrics are necessary for making scheduling decisions. Ahmad et al [14] take this one step further to model the interactions between queries and, using these models, select a mix queries to run concurrently that minimizes contention in the system. This work takes advantages of the unique characteristics of report generation workloads and enforces a fixed multi-programming level (MPL). In contrast, our approach allows the MPL to vary during workload execution and allows for a general workload mix.

Like the work of Ahmad et al, scheduling approaches to control DBMS workloads often control the multi-programming levels; that is, workload control is achieved by controlling the number of queries running concurrently in the system. The work by Schroeder et al. [12] uses queuing theoretic models and a feedback control loop to predict the relationship between throughput, response time and multi-programming levels to optimize the MPL. Although Schroeder et al. evaluate this approach using query priorities in which high priority queries should be chosen to run first, it is also relevant in terms of scheduling for resource control. If the queue is larger, then a query with resource requirements suitable to the currently available resources is more likely to be found. Mehta et al. [13] focus on scheduling business intelligence (BI) batch workloads and attempt to optimize overall response time for the workload. Queries are admitted based on their priority and memory requirements.

Our approach uses models based on information from the optimizer to predict the CPU, I/O and Sort Heap memory required by individual queries. These resources are considered "high impact" resources in a DBMS. We use these predicted measures along with scheduling algorithms to choose which queries will be allowed to run concurrently in the system so as to make efficient use of the system resources and avoid resource overload. Our work is distinctive in that we are considering multiple resources in scheduling decisions.

III. ARCHITECTURE AND APPROACH

Our system can be considered a "load control system", that is, one which controls the current workload executing on the DBMS. The architecture of the load control system is shown in Figure 1. Clients submit queries to the DBMS which are intercepted by the scheduler which consults the DBMS to collect pertinent information regarding potential resource usage. Using this information, a prediction is made by the Resource Requirements Estimator for CPU and I/O usage and the memory requirements for the sort heap. The query is then queued for admission. The Requirements Model contains the policies that rule how the scheduling decisions are made. The Scheduler constantly checks the queue and, if a query can be admitted into the system based on its requirements and the current state of the system, then the query is allowed to proceed. We outline the various components in more detail in the following sections.

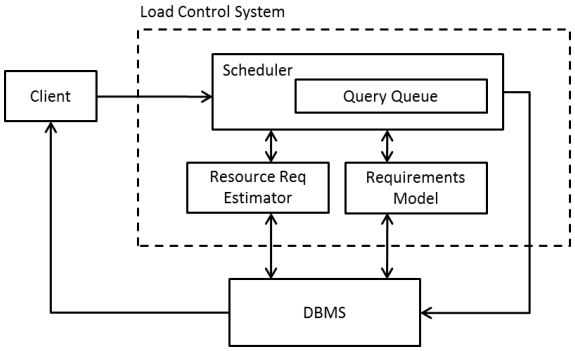

Figure 1. Prototype Architecture

A. Requirements Estimator

The requirements estimator predicts the amount of CPU, I/O and sort memory that will be used by a query. The system uses these estimates along with an estimate of the current resource usage to determine whether or not the query can be admitted to the system at a specific point in time.

Estimates are derived from statistics provided by the DB2 Explain tool [1] which generates an access plan for a given query complete with statistics pertaining to the cost of execution of the plan. Relevant statistics for our work include the cumulative CPU cost (measured in the number of instructions required to execute the query), the cumulative I/O cost (the total number of seeks and page transfers executed by the query), the total cost (a weighted sum of I/O and CPU costs for the query expressed in a measurement that IBM calls "*timerons"*) and sort-related costs such as the number of rows to be sorted and the approximate length of each row. Details of how the estimates for each resource are derived are described below.

CPU

The CPU is at maximum capacity when it is nearing 100% utilization. DB2 Explain provides an estimate of the *cumulative* CPU cost (in number of instructions) of executing a particular query plan. The cumulative estimate is the total amount of CPU that will be used during query execution. For our estimates, it is more useful to know the average amount of CPU that will be used over the lifetime of the query. To estimate the *average* CPU cost during the query execution, we divide the cumulative CPU cost by the overall cost of the query provided by DB2 Explain. To estimate the average CPU cost for each individual query, we ran each of our experimental queries (of which there were 17) alone (without competing workload) 100 times while noting both the estimated and the actual CPU cost (average percentage CPU used during query execution). We have found that there is a relatively high correlation ($r = 0.7$, $n = 1700$, $p = 0.05$) between the estimated value and the actual

 ISBN: 978-1-61208-331-5

measured average CPU. We used linear regression to find a formula to predict the percent utilization of CPU for a query given the estimates from DB2 Explain tool. The equation used was:

%CPU = (0.0001 * cumulative_cpu/overall_cost) + 8.39 (1)

I/O

To determine when the I/O subsystem was nearing capacity, we measured the maximum throughput for our I/O channel using a large database table scan with a small buffer pool. The maximum observed throughput was approximately 190MB/s. A reasonable (r = 0.65, n=1700, p = 0.05) correlation exists between the cumulative number of I/Os predicted by the DB2 Explain facility and the average measured throughput of each query run alone. To calculate this correlation, each of the 17 queries was run alone (without competing workload) 100 times while measuring the average measured throughput. Linear regression was used to derive a formula to estimate the average throughput for a query as follows:

Throughput(MB/s) = (0.00004 * cumulative_io) + 19.2 (2)

Sort Heap

The amount of sort heap memory allocated in a DBMS is important to performance because extending the sort heap leads to spills to disk requiring additional I/O and increased response times.

The estimation of the sort heap required by a query, like for CPU and I/O estimation, uses the information contained in the query execution plan. The plan consists of nodes in a tree structure with each non-leaf node representing an operator. Two DB2 operators require sort heap space; sort and hash join. The amount of sort heap required by each of these operators is determined by the query execution plan which provides the number of rows to be sorted as well as the approximate width of each row in bytes. We experimentally determined that there is approximately 75 bytes per row of overhead. Therefore, the estimate for a single sort operator is

RequiredSortMemory = #Rows * (RowWidth + 75) (3)

The DB2 *sortheap* parameter limits the amount of sort heap space that can be assigned to a single sort or hash join. Therefore, the minimum of the value of the sortheap parameter or RequiredSortMemory is used as the estimate for the sort requirements.

Given that not all nodes in a plan are active at the same time, we cannot simply sum the sort requirements for all the nodes in a query [4]. We determine which nodes can be active at the same time by the types of nodes (blocking, not blocking) and the relationships between them (ancestor, descendant). The sort heap estimation process for a complete query plan can be separated into two steps: calculating sort heap sets and using the sort heap sets to calculate sort heap requirements. Both sorts and hash joins are blocking operations. Hence, any node that requires sort heap is a blocking node. This means that when node N becomes active, the sort heap demand is constant for a period of time, until N starts to produce output. Specifically, the amount of sort heap required while N is blocking is the amount that N requires plus that which its active descendants require. This total amount of sort heap is referred to as the sort heap set of N. Conceptually, a sort heap set for node N is calculated by starting at N and traversing towards the leaves of the query execution tree, summing the sort heap requirements of the traversed nodes, until blocking nodes are encountered.

The amount of sort heap required varies throughout its execution. In other work, we evaluate different estimations including the average usage, the dominant usage and the maximum usage [2]. In the current work, we use the average estimate, that is, the average amount of sort heap that a query will use during its execution time.

B. Requirements Model

The Requirements Model represents the current resource status of the system, that is, how much of a particular resource is available in total as well as the amount that is in use by currently running work. A model is used for each of the CPU, the I/O subsystem and the sort memory.

For the CPU, we assume that the maximum amount of CPU utilization is 100 percent. Our goal is to keep the resource busy, but not overload it. Our model states that a query "fits" in terms of CPU if the CPU estimate of the current query plus the total sum of the CPU estimates for all currently executing queries is less than or equal to 90 percent and the actual measured CPU usage is less than 100 percent.

The I/O model is based on our measured maximum throughput which was 190MB/s. To avoid overloading the I/O system, we use 185MB/s as our maximum desired throughput.

Our resource estimator provides us with the worst case I/O estimate; that is, all data that is requested will need to be read from disk. In a DBMS, however, recently requested data will often be found in the bufferpool, a main memory cache managed by the DBMS. The data in the bufferpool may be reused by other queries requesting the same data, thus reducing the amount of necessary I/O. To account for data sharing, we measure and incorporate the buffer pool hit rate (expressed as a value between 0 and 100), which is the measure of how often a page access is satisfied without a physical I/O. A hit rate of 50 (percent) means that a requested page is found in the buffer pool 50% of the time. We then calculate the maximum throughput that will be allowed into the system as:

Total_Throughput = 185 + current_hit_rate * 185/100 (4)

The theory is that if the buffer pool hit rate is high, we can allow more work into the system without overloading the disk. If it is low, it means that more physical I/O is occurring, therefore, less work should be allowed into the system.

 ISBN: 978-1-61208-331-5

Our Requirements Model admits a query based on I/O if its estimated I/O plus the sum of the I/O usage of currently running queries is less than Total_Throughput (as calculated above).

A query is admitted into the system in terms of sort heap requirements if the sort heap requirements estimate for the current query plus the sum of the sort heap requirement estimates for currently running queries does not exceed the value specified by the DB2 parameter, *sheapthres_shr*, which limits the total amount of sort heap used by all running queries.

C. Scheduler

The scheduler makes decisions as to which query to run next based on the rules defined by the Requirements Model. We have considered several different scheduling algorithms in previous work [4]. In the current work, we use the First Fit Scheduling algorithm. Queries are queued in the order in which they arrive for execution. The scheduler traverses the queue (from earliest arrival to most recent arrival) and considers the requirements of each query. In order to fit into the system, all conditions must be met for each of the three resources. That is, the query must fit in terms of predicted CP, I/O and sort heap usage in order to be admitted to the system. The first query found that meets all the requirements is admitted to the system for execution.

IV. EXPERIMENTAL EVALUATION

A. Experimental Environment

Our database workload consisted of 17 OLAP queries based on the TPC-H benchmark [3]. The ordering of the queries was randomly assigned (on a per client basis) prior to the run, but was kept constant throughout all subsequent runs. Our OLAP workload was sort-intensive and the queries varied in their use of CPU and I/O.

In order to ensure that the CPU and the I/O subsystems were heavily utilized at some points, we simulated a CPU-intensive workload by running a simple program that consumed approximately 30 percent of the CPU when run alone. We simulated an I/O intensive workload by performing multiple repeat scans on a table not used by our OLAP workload. We used a very small (and separate) bufferpool for the I/O intensive workload to ensure that the I/O subsystem was being used extensively.

Twelve clients each sent the 17 OLAP queries to the system for processing. The workload was varied every two minutes in the following pattern:

1. OLAP workload alone
2. CPU intensive workload + OLAP workload
3. I/O intensive workload + OLAP workload
4. CPU intensive workload + I/O intensive workload + OLAP workload

Each run was repeated 8 times and average values reported. Between each run, the database system was restarted to clear all monitor elements and a sample workload was run to warm up the bufferpool and to bring the database system to a steady state.

DB2 V9.7 was used to house the 3GB database for the OLAP workload. The bufferpool was configured to 1GB. The parameters sortheap (the maximum sort heap allocated to any single query) and sheapthres_shr (the limit on the total amount of sort heap used by all running queries) were set to 500 and 2500 4K pages respectively. The DBMS was run on a dedicated Windows 8 Server machine configured with 8 GB of RAM and a quad core CPU. The clients and the scheduling system were run on a remote machine.

We compared our proposed scheduling approach to a) a system running with no control where queries were run on a first come, first serve basis and, b) to a system where we fixed the maximum multi-programming level (MPL) to four, that is, the maximum number of queries that were allowed to run concurrently was four. This number was determined experimentally to be an optimal setting for steady performance in our configuration [2]. We expected that the scheduling approach would yield better performance than the system running with no control and that it would perform at least as well as when the optimal multiprogramming level was used. We compare our approach with a limited MPL as setting the MPL is a common approach to reducing the amount of resource contention in a database system.

B. Results

The results are summarized in Tables 1, 2 and 3. Table 1 shows general metrics including the total run time for the 204 queries (12 clients each running 17 queries) in minutes (including wait time), the average wait time per query (minutes), the maximum wait time (minutes), the average execution time (minutes) and the maximum multi-programming level (MPL). Table 2 shows CPU Usage and I/O metrics such as the average disk queue length, the maximum disk queue length, the average throughput in MBs per second, and the buffer pool hit rate (percentage). Table 3 presents the sort metrics including the number of post threshold sort operations, the sort overflows, and the number of hash join overflows and small hash join overflows. Sort and hash join overflow operations are an indication of sort heap contention. Overflows occur when not enough memory can be granted to perform a sort in memory. In this case, temporary results are often written to (and re-read from) disk resulting in increased I/O.

TABLE I. GENERAL METRICS

	Total RunTime (mins)	***Average Wait Time (mins)***	***Max Wait Time (mins)***	***Average Execution Time (mins)***	***Max MPL***
No Control	135	0.07	1.1	7.1	12
MPL 4	133	4.7	12.9	7.8	4
First Fit Schedule	127	4.9	15.7	6.8	8

 ISBN: 978-1-61208-331-5

TABLE II. I/O AND CPU METRICS

	Average Disk Queue Length	*Max Disk Queue Length*	*Buffer Pool Hit Rate (%)*	*Average Through-put (MB)*	*Average CPU Usage (%)*
No Control	10.4	43	83	86	94
MPL 4	9.8	42	84	78	94
First Fit Schedule	5.2	30	86	79	91

TABLE III. SORT METRICS

	Post Threshold Sort Operations	*Sort Overflows*	*Hash Join Overflows*	*Small Hash Join Overflows*
No Control	81	116	62	24
MPL 4	77	112	58	21
First Fit Schedule	21	22	20	7

The results show that the overall execution time was reduced by approximately 6% using the scheduling approach over the baseline (no control) or MPL 4 approaches. Although the average wait time per query was higher for the scheduling approach, the average execution time per query was lower, indicating a more efficient use of resources. The load on the I/O subsystem was reduced as indicated by a reduction in the average (and maximum) disk queue length and a lower average throughput. The average CPU usage decreased slightly. Sort operations were improved with significantly fewer post threshold sort operations, sort overflows, hash join overflows and small hash join overflows performed in the scheduling approach than either the baseline or the MPL 4 cases.

V. CONCLUSIONS AND FUTURE DIRECTIONS

We have presented and validated a scheduling approach to DBMS workload control that we plan to incorporate into our framework for autonomic DBMS workload control. The described approach schedules queries based on their predicted resource usage. Based on our experimentation, the approach yields reasonable results and appears to be promising approach for adapting to workload changes.

The current work will be integrated as the scheduling component of a proposed framework for DBMS workload management [10]. This framework provides coordinated control of different workload management techniques such as admission control, execution control, and scheduling. Each component is controlled by a feedback controller which monitors system performance and adjusts the amount of control exerted by the mechanism accordingly. For example, the execution control component consists of a controller that a) determines the type of execution control to use (throttling or query canceling) and b) sets the degree of control (for example, in the case of throttling, the controller would set the amount of throttling based on feedback regarding the system performance). The controller for the scheduler will measure actual system resource usage and will feed this information back to the system to update the requirements estimators, and to set the threshold policies in the requirements models accordingly. Building the autonomic controller for the scheduler and integrating it into our overall workload control framework will be the next step in our work.

We have presented the results of only the "first fit" scheduling algorithm in this paper. Experiments have been conducted with the smallest job first and the blocking query scheduling algorithms that were used in our previous work [4]. Results using these algorithms are similar to those reported here with the main difference being that the average and maximum wait times are vastly increased for longer queries using a smallest job first algorithm.

Currently a query is only allowed to run if it fits in terms of CPU, I/O and sort memory. There are many other variations of this approach which may prove to be useful. The most promising approach currently under investigation is scheduling by "critical resource". That is, the resources are monitored and if the usage of one or more resources enters a pre-determined "critical state", the scheduling algorithm considers only the critical resource(s) when making scheduling decisions. We plan to base this work on work done by Zeldes and Feitelson [11], who present an algorithm for system resource management that focuses on bottleneck resources and allocates them to the most deserving clients.

REFERENCES

[1] IBM DB2 Universal Database. DB2 V9.5 Information Center. Available: http://publib.boulder.ibm.com/infocenter/db2luw/v9r5 [retrieved: Feb 2014].

[2] N. Gruska, Resource-aware Query Scheduling in Database Management Systems. MSC thesis, Queen's University, Kingston, Ontario, July 2011.

[3] Transaction Processing Performance Council, TPC-H Benchmark Specification. Available: http://www.tpc.org/tpch/ [retrieved: Feb 2014].

[4] N. Gruska, W. Powley, P. Martin, P. Bird, and K. McDonald, "Sort-Aware Query Scheduling in Database Management Systems," Proc of 2012 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON 2012), November 2012, pp. 2-10.

[5] P. Martin et al., "The Use of Economic Models to Capture Importance Policy for Autonomic Database Management Systems," Proc. of the 8th Intl. Conf. on Autonomic Computing (ICAC'11) workshops (Autonomic Computing in Economics), June, 2011, pp. 3-10, doi: 10.1145/1998561.1998564

[6] M. Zhang et al., "Utility Function-based Workload Management for DBMSs," Proc of the 7th International Conference on Autonomic and Autonomous Systems (ICAS 2011), May, 2011, pp. 116-121.

 ISBN: 978-1-61208-331-5

[7] W. Powley, P. Martin, M. Zhang, P. Bird, and K. McDonald, Autonomic Workload Execution Control Using Throttling," Proc of the 4th International Workshop on Self-Managing Database Systems (SMDB 2010) in Conjunction with the 26th International Conference on Data Engineering (ICDE 2010), March, 2010, pp. 75-80.

[8] B. Niu, P. Martin, and W. Powley, "Towards Autonomic Workload Management in DBMSs, " Journal of Database Management, 20(3), July - Sept 2009, pp. 1-17.

[9] S. Krompass et al., "Managing Long-Running Queries" In Proc.of EDBT'09, March 2009, pp. 132-143, doi: 10.1145/1516360.1516377.

[10] M. Zhang, P. Martin, W. Powley, P. Bird, and D. Kalmuk, "A Framework for Autonomic Workload Management in DBMSs," Information Technology (special issue on Engineering Adaptive Information Systems), in press.

[11] Y. Zeldes, and D. Feitelson, "On-line Fair Allocations Based on Bottlenecks and Global Priorities," Proc of the 4th ACM/SPEC International Conference on Performance Engineering (ICPE '13), April 2013, pp. 229-240, doi: 10.1145/2479871.2479904.

[12] B. Schroeder, M. Harchol-Balter, A. Iyengar, E. Nahum, and A. Wierman. "How to Determine a Good Multi-Programming Level for External Scheduling," Proc of the 22nd International Conference on Data Engineering, April 2006, pp. 60-66, doi: 10.1109/ICDE.2006.78

[13] A. Mehta, C. Gupta, and U. Dayal, "BI Batch Manager: A System for Managing Batch Workloads on Enterprise Data-warehouses," Proc of the 11th International Conference on Extending Database Technology, March 2008, pp. 640-651, doi: 10.1145/1353343.1353420

[14] M. Ahmad, A. Aboulnaga, S. Babu, and K. Munagala, "Interaction-aware Scheduling of Report-Generation Workloads," The VLDB Journal, 20:589-615, 2011, pp 589-615.

[15] A. Ganapathi, et al, "Predicting Multiple Metrics for Queries: Better Decisions Enabled by Machine Learning," Proc International Conference on Data Engineering (ICDE), March 2009, pp. 592-603, doi: 10.1109/ICDE.2009.130.

[16] A. Silberschatz, P.B. Galvin, G. Gagne, Operating System Concepts, Wiley, 9th Edition, 2012.

 ISBN: 978-1-61208-331-5