Policy for Distributed Self-Organizing Infrastructure Management in Cloud Datacenters

Daniela Loreti, Anna Ciampolini
Department of Computer Science and Engineering, Universitá di Bologna
Bologna, Italy
Email: {daniela.loreti, anna.ciampolini}@unibo.it

***Abstract*—Modern data centers for cloud computing are facing the challenge of an ever growing complexity due to the increasing number of users and their augmenting resource requests. A lot of efforts are now concentrated on providing the cloud infrastructure with autonomic behavior, so that it can take decisions about virtual machine (VM) management across the datacenter's nodes without human intervention. While the major part of these solutions is intrinsically centralized and suffers of scalability and reliability problems, we investigate the possibility to provide the cloud with a decentralized self-organizing behavior. We present a new migration policy suitable for a distributed environment, where hosts can exchange status information with each other according to a predefined protocol. The goal of the policy is twofold: energy saving and load balancing. We tested the policy performance by means of an ad hoc built simulator. As we expected, our distributed implementation cannot perform as good as a centralized management, but it can contribute to augment the degree of scalability of a cloud infrastructure.**

Keywords-Distributed Infrastructure Management; Cloud Computing; Self-Organization; Autonomic Computing

I. INTRODUCTION

The Cloud Computing paradigm experienced a significant diffusion during last few years thanks to its capability of relieving companies of the burden of managing their IT infrastructures. At the same time, the demand for scalable yet efficient and energy-saving cloud architectures makes the Green Computing area stronger, driven by the pressing need for greater computational power and for restraining economical and environmental expenditures.

The challenge of efficiently managing a collection of physical servers avoiding bottlenecks and power waste, is not completely solved by Cloud Computing paradigm, but only partially moved from customers's IT infrastructure to provider's big data centers. Since cloud resources are often managed and offered to customers through a collection of virtual machines (VMs), a lot of efforts concerning the Cloud Computing paradigm are concentrating on finding the best virtual machine (VM) allocation to obtain efficiency without compromising performances.

Since an idle server is demonstrated to consume around 70% of its peak power [1], packing the VMss into the lowest possible number of servers and switching off the idle ones, can lead to a higher rate of power efficiency, but can also cause performance degradation in customers's experience and Service Level Agreements (SLAs) violations.

On the other hand, allocating VMs in a way that the total cloud load is balanced across different nodes will result in a higher service reliability and less SLAs violations, but forces the cloud provider to maintain all the physical machines switched on and, consequently, causes unbearable power consumption and excessive costs.

In addition, we must take into account that such a system is continuously evolving: demand of application services, computational load and storage may quickly increase or decrease during execution. Due to these contrasting targets, the VM management in a Cloud Computing datacenter is intrinsically very complex and can be hardly solved by a human system administrator. For this reason, it is desirable to provide the infrastructure with the ability to operate and react to dynamic changes without human intervention.

The major part of the efforts in this field relays on centralized solutions, in which a particular server in the cloud infrastructure is in charge of collecting information on the whole set of physical hosts, taking decisions about VMs allocation or migration, and operating to apply these changes on the infrastructure [2], [3]. The advantages of these centralized solutions are well known: a single node with complete knowledge of the infrastructure can take better decisions and apply them through a restricted number of migrations and communications. However, scalability and reliability problems of centralized solutions are known as well. Furthermore, as the number of physical servers and VMs grows, solving the allocation problem and finding the optimal solution can be time expensive, so some other approximation algorithm is typically used to reach a sub-optimal solution in a fair computation time [4].

In this work, we investigate the possibility of bringing allocation and migration decisions to a decentralized level allowing the cloud's physical nodes to exchange information about their current VM allocation and self-organize to reach a common reallocation plan. To this purpose, we designed a novel distributed policy, Mobile Worst Fit (MWF), able to both save power (by switching off the underloaded hosts) and keep the load balanced across the remaining nodes as to prevent SLA violations. The policy adopts a decentralized approach: we imagine the datacenter as partitioned into a collection of overlapping neighborhoods, in each of which the local reallocation strategy is applied. Taking advantage from the overlapping, the VM redistribution plan propagates from a local to a global perspective. We analyze the effects of this approach by comparing it with the centralized application of a best fit policy. In particular, we relay on the definition of the Distributed Autonomic Migration (DAM) protocol [5], used

 ISBN: 978-1-61208-331-5

by cloud's physical hosts to communicate and get a common decision as regards the reallocation of VMs, according to a predefined global goal (e.g., power-saving, load balancing, etc.).

We tested our approach by means of DAM-Sim, a software to simulate the behavior of different policies applied in a traditional centralized way or through DAM protocol on a decentralized infrastructure.

The article is organized as follows: in Section II, we show the architectural structure of our system, giving an overview of the DAM protocol and focusing on the adopted MWF policy; in Section III, we show the experimental results obtained by means of the DAM-Sim simulator; Section IV shows the state-of-the-art of Cloud Computing infrastructure management and Section V illustrates our conclusions and future works.

II. Architectural Framework

We present a distributed solution for Cloud Computing infrastructure management, with a special focus on VM migration.

As shown in Fig. 1, the framework is composed of three main layers:

- the infrastructure layer, specifying a software representation of the cloud's entities (e.g., hosts, VMs, etc);
- the coordination layer, implementing the DAM protocol, which defines how physical hosts can exchange their status and coordinate their work;
- the policy layer, containing the rules that every node must follow to decide where to possibly move VMs.

The separation between coordination and policy layer allow us to use the same interaction model with different policies. We describe each layer in the following sections.

A. Infrastructure Layer

The infrastructure layer defines which information must be collected about each host's status. To this purpose two basic structures are maintained: the *HostDescriptor* and the *VmDescriptor*.

The *HostDescriptor* can be seen as a bin with a certain capacity able to host a number of VMs, each one with a specific request for computational resources. We only take into account the amount of computational power in terms of MIPS offered by each host and requested by a VM. An empty *HostDescriptor* represents an idle server that can therefore be put in a *sleep* mode or switched-off to save power.

The *HostDescriptor* contains not only a collection (the *current map*) of *VmDescriptor*s really allocated on it, but also

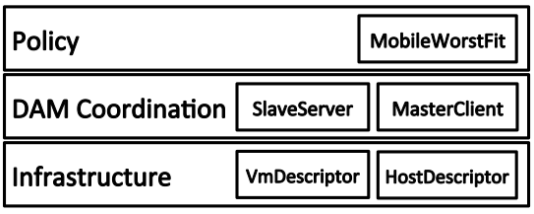

Fig. 1: The three tiers architecture of the Sim-DAM simulator.

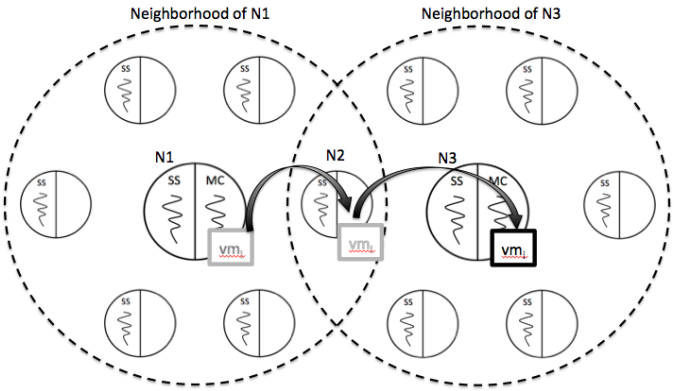

Fig. 2: Schema of two overlapping neighborhoods.

a temporary collection (the *future map*) initialized as a copy of the real one and exchanged between hosts according to the defined protocol. During interactions only the temporary copy is updated and, when the system reaches a common reallocation decision, the *future map* is used to apply the migrations.

In a distributed environment, where each node can be aware only of the state of a local neighborhood of nodes, the number of worthless migrations can be very high. Thus, this double-map mechanism is used to limit the number of migrations (as we describe in Section II-B), by performing them only when all the hosts reach a common distributed decision.

Each VM is also equipped with a migration history keeping track of all the hosts where it was previously allocated. For the sake of simplicity, we assume that a VM cannot change its CPU request during the simulation period.

B. Coordination Layer

The coordination layer implements the DAM protocol which defines the sequence of messages that hosts must exchange in order to get a common migration decision and realize the defined policy.

DAM protocol coordination details are explained in-depth in [5]. The protocol is based on the assumption that the cloud is divided into a predefined fixed collection of overlapping subsets of hosts: we call each subset a *neighborhood*.

We assume that each physical host executes a daemon process called *SlaveServer* (SS in Fig. 2), which owns a copy of the node's status stored into an *HostDescriptor* and can send it to other nodes asking for that.

Each node can monitor its computational load and the amount of resources used by the hosted VMs; according to the chosen policy, it can detect either it is in a critical condition or not. A node can, for example, detect to be overloaded, risking to incur in SLA's violations, or underloaded, causing possibile power waste. If one of these critical conditions happens, the node starts another process, the *MasterClient*, to actually make a protocol interaction begin. We call *rising condition* the one that turns on a node's *MasterClient*.

Since there is a certain rate of overlapping between neighborhoods, the effects of migrations within a neighborhood can cause new rising conditions in adjacent ones.

 ISBN: 978-1-61208-331-5

To better explain the DAM protocol, Fig. 2 shows an example of two overlapping neighborhoods. Each node has a *SlaveServer* (SS in Fig. 2) always running to answer questions from other node's *MasterClient* (MC in Fig. 2) , and optionally can also have a *MasterClient* process started to handle a critical situation. A virtual machine vm allocated to an underloaded node N1 can be moved out of it on N2 and, as a consequence of the execution of the protocol in the adjacent neighborhood of N3, it can be moved again from N2 to N3. It is worth to notice that node N2, as each node of the datacenter, has its own fixed neighborhood, but it starts to interact with it (by means of a *MasterClient*) only if a *rising condition* is observed.

Note that N1's *MasterClient* must have N2 in its neighborhood to interact with it, but the *SlaveServer* of N2 can answer to requests by any *MasterClient* and, if a critical situation is detected (so that N2 *MasterClient* is started) its neighborhood does not necessarily include N1.

As regards this environment, we must remark that the migration policy should be properly implemented in order to prevent never-ending cycles in the migration process.

We must ensure that the neighbors's states the *MasterClient* obtains, are consistent from the beginning to the end of the interaction. For this reason, a two-phase protocol is adopted:

1) DAM Phase 1: The *MasterClient* sends a message to all the *SlaveServers* neighbors to collect their *HostDescriptors*. This message also works as a *lock* message: when the *SlaveServer* receives it, locks his state, so that no interactions with other *MasterClients* can take place. If a *MasterClient* sends a request to a locked *SlaveServer*, simply waits for the *SlaveServer* to be *unlocked* and to send its state.

2) DAM Phase 2: The *MasterClient* compares all the received neighbor's *HostDescriptors* with a previous copy he stored. If the *future map* allocation is changed, performs phase 2A, otherwise increments a counter and, when it exceeds a certain maximum, performs phase 2B:

- *Phase 2A:* the *MasterClient* computes a VM reallocation plan for the whole neighborhood, according to the defined policy, and sends back to each *SlaveServer* neighbor the modified *HostDescriptor*. The state is accepted passively by the slaves, without contradictory. The migration decisions only change the *future map* of VM allocation. No host switch-on/off or VM migration is performed in this phase. After all new states are sent, the *SlaveServers* are *unlocked* and the *MasterClient* begins another round of the protocol interaction by restarting phase 1.
- *Phase 2B:* when the number of round with unchanged neighbor's allocation exceeds a defined maximum, the *MasterClient* sends an *update-current-status* request to all *SlaveServers* and terminates. This last message notifies the *SlaveServers* that information inside the *HostDescriptor* should be applied to the real system state. The *SlaveServer* again executes it passively and unlocks his state.

Phase 2A and 2B alternatives come from the need for reducing the number of migration physically performed. Looking at example in Fig. 2, if hosts only exchange and update the current collection of VMs, every *MasterClient* can only order a real migration at each round, so that vm_i on N1 would be migrated on N2 at first, and later on N3. Using the temporary *future map* (initially copied from the real one) and performing all the reallocations on this abstract copy, real migration are executed only when the N3's *MasterClient* exceeds a maximum number of rounds and vm_i can directly go from N1 to N3.

Input: h, t, FTH_DOWN, FTH_UP.

```
1:  ave = calculateNeighAverage();
2:  MTH_DOWN = ave − t;
3:  MTH_UP = ave + t;
4:  u = h.getLoad();
5:  if u < FTH_DOWN or u < MTH_DOWN then
6:     vmList = h.getFutureVmMap();
7:  else if u > FTH_UP or u > MTH_UP then
8:     vmList = selectVms();
9:  end if
10: if vmList.size ≠ 0 then
11:    migrateAll(vmList);
12: end if
```

Fig. 3: MWF policy algorithm.

C. Policy Layer

The Policy Layer is responsible for the decentralized migration decision process. This paper presents MWF, a novel policy aiming to switch off the underloaded hosts to save power, while maintaining the load of the other nodes balanced. MWF exploits two fixed thresholds (FTH_UP and FTH_DOWN) and two dynamic thresholds (MTH_UP and MTH_DOWN) used to detect rising conditions. The fixed thresholds identify risky situations: if the host is less loaded than FTH_DOWN an energy waste is detected, while, if the host is more loaded than FTH_UP, SLA violations may occur. The dynamic thresholds (MTH_UP and MTH_DOWN) represents the upper and lower values that cannot be exceeded in order to maintain the neighborhood balanced.

According to the DAM coordination protocol, at each iteration the *MasterClient* collects the VM allocation map of the neighbors and executes a MWF optimization as detailed in Fig. 3: the *MasterClient* calculates the average of resource utilization in his neighborhood (*calculateNeighAverage()* in line 1 of Fig. 3) and uses it to compute the two dynamic thresholds (MTH_DOWN and MTH_UP) by adding and subtracting a tolerance interval t (lines 2-3 of Fig.3). Then the *MasterClient* checks its *HostDescriptor* h and collects the current computational load u by invoking a specific $getLoad()$ method on the *HostDescriptor* (line 4 of Fig. 3).

The computational load u of the host is compared to fixed and dynamic thresholds: if it is less than the lower thresholds, the *MasterClient* attempts to put the host in *sleep* mode by migrating all the VMs allocated; otherwise, if the host load exceeds the upper thresholds, only a small number of VMs are selected for migration. As we can see in line 5-6 of Fig.3, if the computational load u is less than the fixed (FTH_DOWN)

 ISBN: 978-1-61208-331-5

Input: h, MTH_UP, FTH_UP. **Output:** vmsToMove.

```
1:  u = h.getLoad();
2:  vmList = h.getFutureVmMap();
3:  vmList.sortDecreasingLoad();
4:  minU = ∞; bestVm = null;
5:  thr = min{FTH_UP, MTH_UP};
6:  vmsToMove = emptyList();
7:  while u > thr do
8:    for each vm in vmList do
9:      var = vm.getLoad() − u + thr;
10:     if var ≥ 0 then
11:       if var < minU then
12:         minU = var;
13:         bestVm = vm;
14:       end if
15:     else
16:       if minU == ∞ then
17:         bestVm = vm;
18:       end if
19:       break;
20:     end if
21:   end for
22:   u = u − bestVm.getLoad();
23:   vmsToMove.add(bestVm);
24:   vmList.remove(bestVm);
25: end while
```

Fig. 4: The selectVms() procedure.

or the dynamic (MTH_DOWN) lower thresholds, all the VMs of the host are collected for migration into an array $vmList$. $h.getFutureVmMap()$ in line 6 is the method to collect the temporary allocation. Indeed in this phase, the policy only works on a copy of the real VM allocation map, because according to DAM protocol, all the migrations will be performed only when the whole datacenter reach a common decision. If the load u is detected to be higher than the fixed (FTH_UP) or dynamic (MTH_UP) upper thresholds, then the $selectVm()$ operation is invoked to pick (from the host h temporary state) only the less loaded VMs whose migration will result in the host load to go back under both MTH_UP and FTH_UP. $selectVm()$ is a modified version of Minimization of Migrations algorithm from Beloglazov et al. [6] and is detailed in Fig. 4. Differently from [6], we select the threshold thr as the minimum between FTH_UP and MTH_UP.

The list of chosen VMs $vmList$ is finally migrated to neighbors by means of a modified worst-fit policy ($migrateAll(vmList)$ in line 11 of 3). As shown in Fig. 5, the $migrateAll$ procedure takes as input the list of vm to move ($vmList$), the host h where they are currently allocated, the list $offNeighList$ of switched-off hosts in h's neighborhood, the $underNeighList$ of h's neighbors with load level lower than FTH_DOWN, and $otherNeighList$ of all the other neighbors of h. The procedure considers the VMs by decreasing CPU request and, according to the principles of worst-fit algorithm, tries to migrate it to the neighbor n with the highest value of free capacity (lines 2-13 of Fig. 5). If no neighbor in $otherNeighList$ can receive the vm, the $underNeighList$ is considered with a best-fit

Input: vmList, h, offNeighList, underNeighList, otherNeighList.

```
1:  vmList.sortDecreasingLoad();
2:  for each vm in vmList do
3:    vmU = vm.getLoad();
4:    maxAvail = 0; bestHost = null;
5:    for each n in otherNeighList do
6:      if n ∉ vm.getMigrationHistory() then
7:        avail = FTH_UP − n.getLoad() + vmU;
8:        if avail > maxAvail then
9:          maxAvail = avail;
10:         bestHost = n;
11:       end if
12:     end if
13:   end for
14:   if bestHost == null then
15:     minU = ∞
16:     for each n in underNeighList do
17:       if n ∉ vm.getMigrationHistory() then
18:         avail = FTH_UP − n.getLoad() + vmU;
19:         if avail >= 0 and avail < minU then
20:           minU = avail;
21:           bestHost = n;
22:         end if
23:       end if
24:     end for
25:   end if
26:   if bestHost == null and !empty(offNeighList)
      and u > FTH_UP then
27:     bestHost = offNeighList.get(0);
28:   else
29:     migrationMap = null; {all-or-none behavior}
30:     break;
31:   end if
32:   migrationMap.add(vm, bestHost);
33: end for
34: commitOnFutureMap(migrationMap);
```

Fig. 5: The migrateAll() procedure.

approach (lines 14-25 of Fig. 5), thus allocating vm on the most loaded host of the list. This ensure that neighbors with CPU utilization near to FTH_DOWN are preferred, while less loaded ones remain unchanged and will be hopefully switched-off by other protocol's interactions. Finally, if neither hosts in $underNeighList$ can receive vm (e.g, because the list is empty), but h is more loaded than FTH_UP, then h is in a risky situation because SLA's violations can occur. Thus a switched-off neighbor is woken up (line 27 of Fig. 5). $migrateAll(vmList)$ operates in a "all-or-none" way, such that the migrations are committed on the future maps (line 34 of Fig. 5) only if it is possible to reallocate all the VMs in the list (i.e., without making other hosts to exceed FTH_UP), otherwise no action is performed (line 29 of Fig. 5).

As shown in Fig. 6, suppose that a protocol execution by the *MasterClient* of h_b decides to migrate a virtual machine vm_i currently allocated on h_c to h_b. When the *SlaveServer* of h_b is unlocked, the policy execution on h_a's *MasterClient* can decide to put vm_i into h_a. Now if h_c has a *MasterClient*

 ISBN: 978-1-61208-331-5

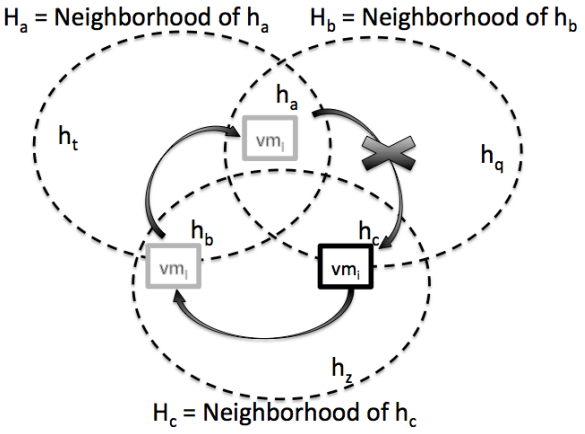

Fig. 6: Example of three overlapping neighborhoods.

running, and decides to migrate vm_i back to h_c, then h_c can take the same decision as before and a loop in vm_i migration starts. If this happens, the distributed system will never converge to a common decision. In order to face this problem, the MWF policy exploits the migration history inside each *VmDescriptor* to avoid loops in reallocation: a VM can be migrated only on a host that it never visited before. Once the distributed autonomic infrastructure reached a common decision, the migration history of each VM is deleted.

III. Experimental results

To understand the effectiveness of the proposed model we developed DAM-Sim [5]: a Java simulator able to apply a specific policy on a collection of neighborhoods through DAM protocol and compare the performance with a centralized policy implementation.

We tested our approach on a set of 100 physical nodes hosting around 3000 VMs (i.e., an average value of 30 VMs on each host), repeating every experiment with an increasing average load on each physical server. We fixed the FTH_DOWN threshold at 25% of computational load and the FTH_UP at 95%, while the tolerance interval t for load balancing is fixed at 8%. We always start from the worst situation for power-saving purposes, i.e., all the servers are switched on and have the same computational load within the fixed thresholds. To make the DAM protocol start we need some lack of balance in the datacenter, so we forced 20 hosts to be more loaded and 20 hosts to be less loaded than the datacenter average value.

In Fig. 7, we compare the MWF performance with nN=5 and 10 nodes in each neighborhood, with the application of a centralized best fit policy (GLO in Fig. 7a). We also show the performance of a best fit policy applied in a distributed way by means of DAM protocol (BF in Fig. 7d). Details about BF implementation can be found in [5].

Fig. 7a and 7d show the number of servers switched on at the end of the MWF and BF executions. As we expected, the DAM protocol cannot perform better than a global algorithm. Indeed, the global best fit policy can always switch off a higher rate of servers resulting in the lower trend. Furthermore, as regards the power saving objective, we can see that BF perform better than MWF for all the selected neighborhood dimensions. This comes from the different objectives of the two policies: MWF tries to switch-off the initially underloaded servers to save power, while keeping the load of the working servers balanced; BF brings into question all the neighborhood allocation at each *MasterClient* interaction, considering only power-saving objectives.

Fig. 7b and 7e show the number of migrations executed. Since the number of VMs can vary a bit from a scenario to another and the number of switched off servers influences the result, in the graph we show the following rate:

$$nMig\frac{onServers}{nVM} \tag{1}$$

where $nMig$ is the number of migrations performed, $onServers$ is the number of working servers at the end of the simulation and nVM is the number of VMs in the initial scenario.

Since no information about the current allocation of a VM is taken into account during the policy computation in a global environment, the number of migrations can be very high. Indeed is high the resulting trend of migration for the global policy, while DAM always outperforms it. In particular, MWF performs better than BF for every selected neighborhood dimension. Nevertheless, for high value of computational load the performance of MWF in terms of number of switched off server are comparable to those of the global best fit policy, while the migration rate is significantly lower.

Fig. 7c and 7f show the number of messages exchanged between hosts during the computation. As we expected, it significantly increases as the number of servers in each neighborhood grows. Even if the number of messages for low values of neighborhood dimension is comparable to the one of the global solution, when it grows, the number of messages exchanged significantly increases.

At the moment, the simulator is not able to give trustworthy results about execution time for distributed environments, because the CPU executing the simulator code can only sequentialize intrinsically concurrent processes of the protocol. For this reason, no test about execution time is reported.

In Fig. 8, we can see the distribution of number of servers along load intervals. In the initial scenario (INITIAL in Fig. 8) all the servers have 50% load except for 20 underloaded and 20 overloaded nodes. We show the distribution after a global best fit optimization (GLO in Fig. 8) and the application of MWF and BF by means of DAM protocol with 5 as neighborhood dimension.

The application of a global best fit switches-off a large number of servers to save power, but packs too much VMs on the remaining hosts. This results in the red distribution in Fig. 8, where almost all the switched on servers are loaded at 95s% creating an high risk of SLAs violations. The best fit (BF) algorithm applied by means of DAM protocol suffers of the same problem: a large number of servers is switched-off, but a part is forced to have 95% load. MWF is more

 ISBN: 978-1-61208-331-5

effective from the load balancing perspective: it can switch-off less servers than BF, but is able to decrease the load of the overloaded nodes leaving all the working servers balanced.

As we expected, Fig. 8 reveals that the median of the MWF distribution is augmented respect to the initial configuration. This is due to the fact that a certain number of servers is switched-off, thus the global load of the remaining servers results increased.

IV. Related works

Our work mainly concern low level infrastructural support, in which the management of virtualized resources is always a compromise between system performance and energy-saving. Indeed, in a cloud infrastructure there are usually well-defined SLAs to be compliant to and perhaps the simplest solution is to use all the machines in the cloud. Nevertheless, if all the hosts of the datacenter are switched on, the energy waste increases leading to probably too high costs for the cloud provider.

Around cloud environments, with their contrasting targets of energy-saving versus performance and SLAs compliance, a lot of work was done in order to provide some kind of autonomy from human system administration and reduce complexity. Some of these works involves automatic control theory realizing an intrinsic centralized environment, in which the rate of utilization of each host is sent to a collector node able to determine which physical machines must be switched off or turned on [2], [3], [7]. Some other solutions concern centralized energy-aware optimization algorithms [4], [8], [9], in particular extensions of the Bin Packing Problem [10], [11] to solve both VMs allocation and migration problems [6]. These approaches focus on finding the best solution and minimizing the complexity of the algorithm, without concerning the particular implementation, but assuming a solver aware of the whole system state (in terms of load on each physical host and VM allocation). Thus they particularly lend to a centralized implementation.

Finally, other approaches involve intelligent, optionally bio-inspired [12], [13], agent-based system, which can give to the datacenter a certain rate of independence from human administration, showing an intelligent self-organizing emergent behavior [14], [15], and also provide the benefits of a more distributed system structure [16]. As in [14] which is based on Gossip protocol [17], we adopt a self-organizing approach, where coordination of nodes in small overlapping neighborhoods leads to a global reallocation of VMs, but differently from [14] we created a more elaborate model of communication between physical hosts of the datacenter. In particular, while in [14] each migration decision is taken after a peer-to-peer interaction comparing the states of the only two hosts involved, in our approach the migration decisions are more accurate because they comes from an evaluation of the whole neighborhood state.

V. Conclusions

We presented a VM migration policy suitable for a distributed management in a cloud datacenter. To do so, we relayed on a decentralized solution for cloud virtual infrastructure management (DAM), in which the hosts of the datacenter are able to self-organize and reach a global VM reallocation plan, according to a given policy.

We tested the policy behavior by means of a software simulator. MWF shows good performances for various computational loads in terms of both number of migrations requested and number of switched-off servers. MWF is also able to achieve an appreciable load balancing among the working servers, while still some work remain to do to decrease the number of messages exchanged. Therefore in the near future, we plan to optimize the DAM protocol in order to reduce the amount of messages in each interaction. As we expected, the distributed MWF policy cannot outperform a centralized global best-fit policy (especially in terms of number of switched-off hosts and exchanged messages), but the decentralized nature of our approach can intrinsically contribute to augment the scalability of the cloud management infrastructure.

In the near future, we will use DAM-Sim to test different and more elaborate reallocation policies, taking into account not only computational resources, but also memory and bandwidth requirements. We will introduce variations of VM load requests at simulation time to better mirror real datacenter environments. Furthermore, in this work, we avoid loops in VM migrations by preventing the allocation on nodes that already hosted the same vm before. We plan to relax this restrictive constraint by means of a Most Recently Used queue of hosts.

Finally, we plan to test our implementation on a real cloud infrastructure and compare the time to get a common distributed decision with the centralized implementation of the same reallocation policy. Furthermore, on a real cloud infrastructure we expect to face low level architectural constraints in overlapping neighborhoods definition, which will request deeper investigations.

References

[1] X. Fan, W.-D. Weber, and L. A. Barroso, “Power provisioning for a warehouse-sized computer,” in The 34th ACM International Symposium on Computer Architecture. ACM New York, 2007, pp. 13–23.

[2] Jung, “Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures,” in International Conference on Distributed Computing Systems, IEEE, Ed., June 2010, pp. 62–73.

[3] H. C. Lim, S. Babu, and J. S. Chase, “Automated control in cloud computing challeges and opportunities,” in ACDC ’09, Proceedings of the 1st workshop on Automated control for datacenters and clouds. ACM New York, 2009, pp. 13–18.

[4] A. Beloglazov and R. Buyya, “Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers,” Concurrency and Computation: Practice and Experience, vol. 24, no. 13, pp. 1397–1420, September 2012.

[5] D. Loreti and A. Ciampolini, “Green-dam: a power-aware self-organizing approach for cloud infrastructure management,” Università di Bologna, Tech. Rep., 2013 - http://www.lia.deis.unibo.it/Staff/DanielaLoreti/HomePage_files/Green-DAM.pdf.

[6] A. Beloglazov, J. Abawajy, and R. Buyya, “Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing,” Future Generation Computer Systems, vol. 28, no. 5, pp. 755–768, May 2012.

 ISBN: 978-1-61208-331-5

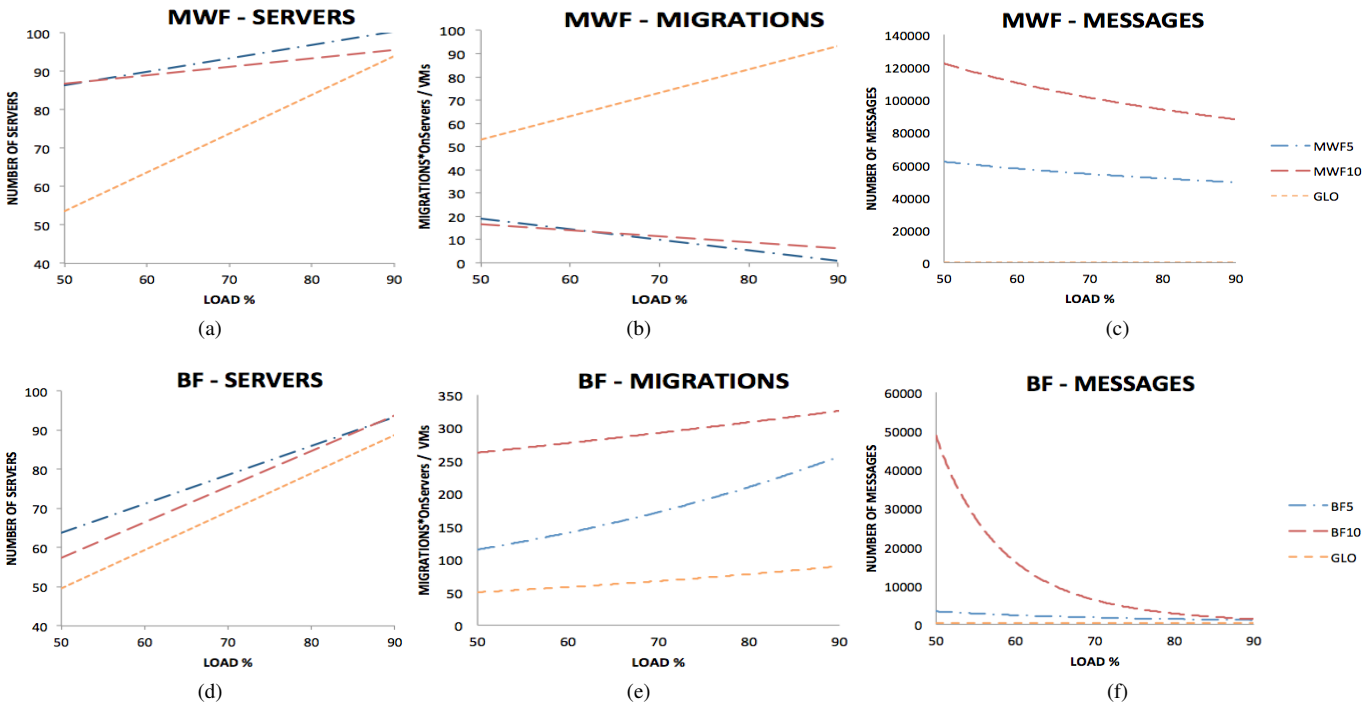

Fig. 7: MWF end BF performance comparison.

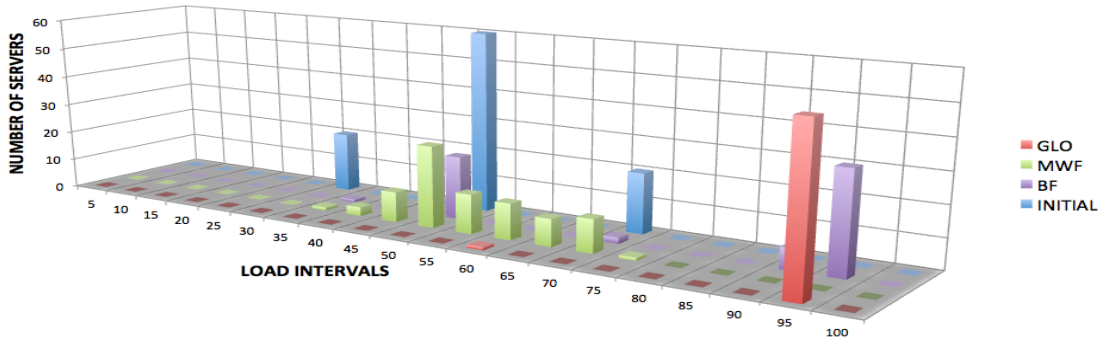

Fig. 8: Distribution of servers on load intervals.

[7] E. Kalyvianaki, "Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters," in ICAC '09 Proceedings of the 6th international conference on Autonomic computing, ACM, Ed., 2009, pp. 117–126.

[8] R. Jansen, "Energy efficient virtual machine allocation in the cloud," in Green Computing Conference and Workshops (IGCC), 2011 International. IEEE, July 2011, pp. 1–8.

[9] A. J. Younge, "Efficient resource management for cloud computing environments," in Green Computing Conference, 2010 International. IEEE, August 2010, pp. 357–364.

[10] J. Levine and F. Ducatelle, "Ant colony optimisation and local search for bin packing and cutting stock problems," Journal of the Operational Research Society, pp. 1–16, 2003.

[11] S. Zaman and D. Grosu, "Combinatorial auction-based allocation of virtual machine instances in clouds," in 2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, Ed., December 2010, pp. 127–134.

[12] R. Giordanelli, C. Mastroianni, and M. Meo, "Bio-inspired p2p systems: The case of multidimensional overlay," ACM Transactions on Autonomous and Adaptive Systems (TAAS), vol. 7, no. 4, p. Article No. 35, December 2012.

[13] S. Balasubramaniam, K. Barrett, W. Donnelly, and S. V. D. Meer, "Bio-inspired policy based management (biopbm) for autonomic communications systems," in 7th IEEE International workshop on Policies for Distributed Systems and Networks, IEEE, Ed., June 2006, pp. 3–12.

[14] M. Marzolla, O. Babaoglu, and F. Panzieri, "Server consolidation in clouds through gossiping," Technical Report UBLCS-2011-01, 2011.

[15] A. Vichos, "Agent-based management of virtual machines for cloud infrastructure," Ph.D. dissertation, School of Informatics, University of Edinburgh, 2011.

[16] M. Tighe, G. Keller, M. Bauer, and H. Lutfiyya, "A distributed approach to dynamic vm management," in Network and Service Management (CNSM), 2013 9th International Conference on, October 2013, pp. 166–170.

[17] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dinamic networks," ACM Transaction on Computer Systems, vol. 23, no. 3, pp. 219–252, August 2005.

 ISBN: 978-1-61208-331-5