A Decentralized Approach for Virtual Infrastructure Management in Cloud Datacenters

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

***Abstract*—The ever growing complexity of modern data centers for cloud computing - mainly due to the increasing number of users and their augmenting requests for resources - is pushing the need for new approaches to cloud infrastructure management. In order to face this new complexity challenge, many organizations have been exploring the possibility of providing the cloud infrastructure with an autonomic behavior, i.e., the ability to take decisions about virtual machine (VM) management across the datacenter's physical nodes without human intervention. While many of these solutions are intrinsically centralized and suffer of scalability and reliability problems, we investigate the possibility to provide the cloud with a decentralized self-organizing behavior. We present a novel migration policy with a twofold goal: saving energy (by putting in sleep mode the underutilized nodes of the datacenter), while keeping the load balanced across the working physical machines. Our migration policy is suitable for a distributed environment, where hosts can exchange status information with each other according to a predefined protocol. We evaluate the performance of the approach by means of an ad hoc built simulator. As we expected, although our distributed implementation cannot perform as good as a centralized management, it can significantly contribute to augment the degree of scalability of a cloud infrastructure.**

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

I. INTRODUCTION

This article is an extended version of the work [1] presented at ICAS 2014. It reports more detailed explanation of the model, as well as further investigation of the performance of the approach in a simulated senario.

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 [2], 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.

The operation of turning back on a previously switched off host can be very time-consuming. In modern data centers, aiming to obtain a more reactive system, the underloaded hosts are never completely switched off, but only put into sleep mode. This technique slightly increases the power consumption, but also speeds up the wake up process when other computational power is needed.

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 relies 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 [3], [4]. 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 [5]. The adoption of a centralized VM management is even unfeasible in those contexts (like Community Cloud [6], [7]

and Social Cloud Computing [8]), in which both the demand for computational power and the amount of offered resources can change dynamically.

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 rely on the definition of the Distributed Autonomic Migration (DAM) protocol [9], used 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 that simulates 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 report the experimental results obtained by means of the DAM-Sim simulator; Section IV focuses on the state-of-the-art of Cloud Computing infrastructure management and Section V describe our conclusions and future works.

II. The Architecture

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

As shown in Fig. 1, each physical node of the system is equipped with a software module 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. In this way, different goals can be achieved by only changing the adopted policy, while the communication model remains the same. We describe each layer in more detail in the following sections.

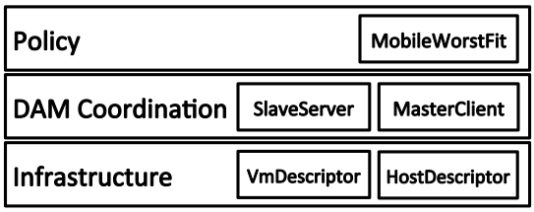


Fig. 1: The three tiers architecture. The separation between layers ensures the possibility to test different policies and protocols with the same infrastructure implementation.

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 given 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 of *VmDescriptor*s really allocated on it (the *current map*), but also 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.

1) The CPU model: The amount U_h of CPU MIPS used by the host h is calculated as follows:

$$U_h = \sum_{vm \in currentVmMap_h} m_{vm} \frac{T_{vm}}{100} \qquad (1)$$

where $currentVmMap_h$ is the set of virtual machines currently allocated on host h; T_{vm} is the total CPU MIPS that a virtual machine vm can request; and m_{vm} is the percentage of this total that is currently used.

Indeed, we consider a simplified model in which the total MIPS executed by the node can be seen as the sum of

MIPS used by each hosted virtual machine. In fact, the model does not take into account the power consumed by the physical machines to realize virtualization and to manage their resources.

B. Coordination Layer

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

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*. From an operational point of view, we define a *"knows the neighbor"* relation between the hosts of the datacenter, which allow us to partition the cloud into overlapping neighborhoods of physical machines. As we can see in Fig. 2 the relation is not symmetric.

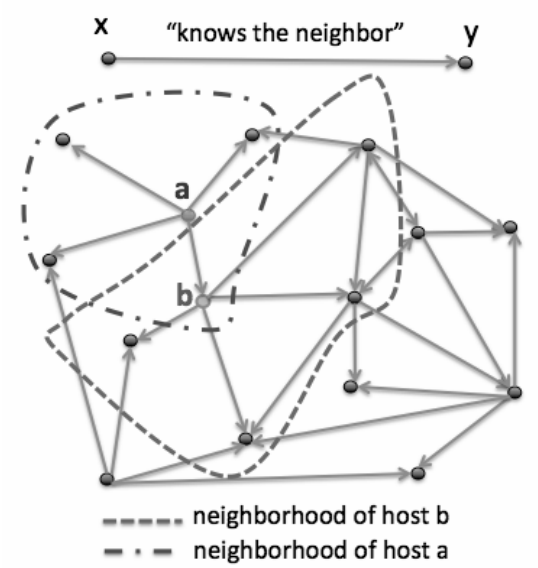


Fig. 2: Example of *"knows the neighbor"* relation applied on a collection of physical nodes. The relation is not symmetric, thus if node *"a knows the neighbor b"*, this does not mean that b is included in the neighborhood of a but, in general, a is not in the neighborhood of b.

We assume that each physical host executes a daemon process called *SlaveServer*, 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*.

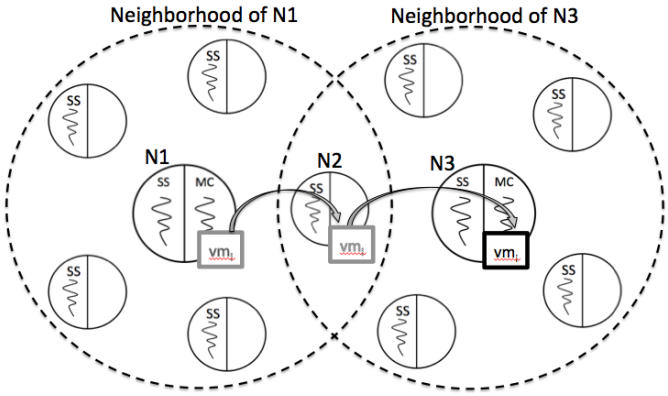


Fig. 3: Schema of two overlapping neighborhoods. The VM descriptor vm_i is exchanged across physical hosts, crossing the neighborhood boundaries, until the nodes agree with a common reallocation plan i.e., a "stable" allocation hypothesis for vm_i is detected.

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.

To better explain the DAM protocol, Fig. 3 shows an example of two overlapping neighborhoods. Each node has a *SlaveServer* (SS in Fig. 3) always running to answer questions from other node's *MasterClient* (MC in Fig. 3), and optionally can also have a *MasterClient* process started to handle a local 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.

Algorithms 1 and 2 reports the interaction code executed by the *MasterClient* and the *SlaveServer*, respectively. The *MasterClient* procedure takes as input the list of *SlaveServer* neighbors *ssNeighList* and an integer parameter *maxRound*. The *SlaveServer* procedure takes the *HostDescriptor h* of the node on which it is running. If the *SlaveServer* detects a critical conditions on the host, makes a *MasterClient* process start (lines 1-2 in Alg. 2).

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: As we can see in lines 6-10 of Alg.1, the *MasterClient* sends a message to all the *SlaveServers* neighbors *ss* to collect their *HostDescriptors h*. This message also works as a *lock* message: when the *SlaveServer* receives it, locks his state, so that no interactions with other *MasterClients*

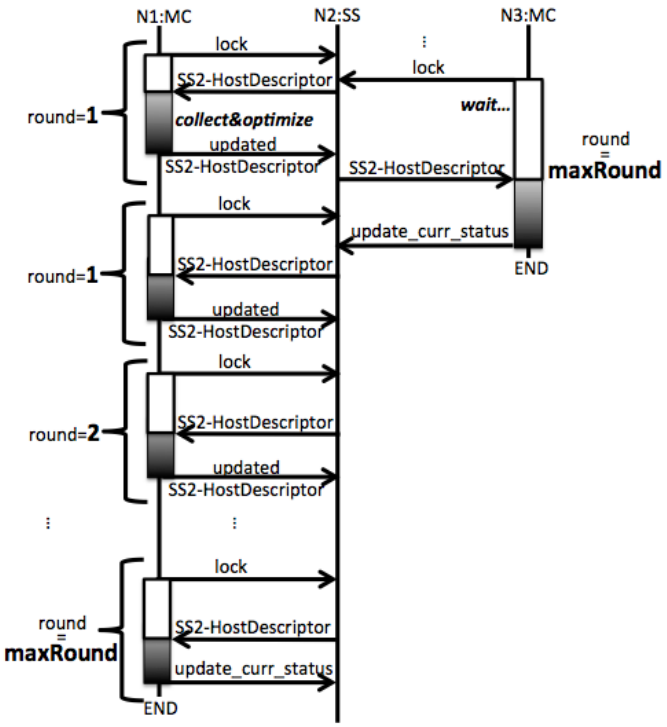


Fig. 4: Example of protocol interaction rounds. Node N2 is shared by nodes N1 and N3. Therefore, their *MasterClients* must coordinate to ensure the consistency of status information.

can take place (lines 6-13 in Alg. 2). 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 (line 12 in Alg. 1). If the *future map* 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* (lines 20-23 in Alg. 1). The "optimize(neighHDs)" operation in line 20 of Alg. 1 actually applies the specific chosen policy on the neighborhood's *HostDescriptors* (neighHDs). Indeed, this method is the software connection between the coordination layer and the policy layer.
  As we can see in line 18 of Algorithm 2, the state is accepted passively by the slaves, without negotiation. 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* (line 23 in Alg. 2) 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 (line 18 of Alg. 1), the *MasterClient* sends an *update-current-status* request (line 27 of Alg. 1) to all *SlaveServers* and terminates. This last message notifies the *SlaveServers* that information inside the *HostDescriptor* should be applied to the real system state (line 21 of Alg. 2). The *SlaveServer* again executes it passively and unlocks his state.

Algorithm 1 MasterClient DAM protocol code

Input: maxRound, ssNeighList

```
1:  neighHDs = emptylist();
2:  neighHDsPast = emptylist();
3:  round = 0;
4:  while true do
5:    {PHASE 1}
6:    for each ss in ssNeighList do
7:      send(ss, "lock");
8:      (h, ss) = receive();
9:      neighHDs.add(h);
10:   end for
11:   {PHASE 2}
12:   if neighHDs == neighHDsPast then
13:     round++;
14:   else
15:     round = 0;
16:     neighHDsPast = neighHDs;
17:   end if
18:   if round < maxRound then
19:     {PHASE 2A}
20:     optimize(neighHDs);
21:     for each ss in ssNeighList do
22:       send(ss, neighHDs.get(ss));
23:     end for
24:   else
25:     {PHASE 2B}
26:     for each ss in ssNeighList do
27:       send(ss, "update_current_status");
28:     end for
29:     break;
30:   end if
31: end while
```

Alternatives 2A and 2B come from the need for reducing the number of migration physically performed. Looking at example in Fig. 3, 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.

The same example is represented in Fig. 4. N2 is shared by the *MasterClient*s of nodes N1 and N3. Two concurrent sessions of the protocol must synchronize in order to maintain the status information consistent. Therefore, node N3 waits until N2 status is updated and released by N1. If no concurrent interactions are taking place in adjacent neighborhoods,

Algorithm 2 SlaveServer DAM protocol code

Input: h

```
1: if checkRisingCondition() then
2:    startMasterClient();
3: end if
4: while true do
5:    {PHASE 1}
6:    (msg, mc) = receive();
7:    if msg! = "lock" then
8:       {protocol error}
9:       break;
10:   else
11:      lock();
12:      send(mc, h);
13:   end if
14:   {PHASE 2}
15:   (item, mc) = receive();
16:   if item! = "update_current_status" then
17:      {PHASE 2A}
18:      h = item;
19:   else
20:      {PHASE 2B}
21:      updateCurrentStatus(h);
22:   end if
23:   unlock();
24:   if checkRisingCondition() then
25:      startMasterClient();
26:   end if
27: end while
```

the *MasterClient* receives an unchanged *HostDescriptor* and increments the value of the *round* counter.

As a result of DAM protocol, the consensus on migration of VMs is not for the entire infrastructure, but is distributed across the neighborhoods. This element must be taken into account while implementing the policy layer.

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 (mobile) 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

Algorithm 3 MWF policy

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
```

Algorithm 4 selectVms() procedure

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
```

Alg. 3: the *MasterClient* calculates the average of resource utilization in his neighborhood (*calculateNeighAverage()* in line 1 of Alg. 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 Alg. 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 Alg. 3).

The computational load u of the host is compared to fixed and dynamic thresholds: if it is less than the lower thresholds,

Algorithm 5 migrateAll() procedure

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;s
31:   end if
32:   migrationMap.add(vm, bestHost);
33: end for
34: commitOnFutureMap(migrationMap);
```

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 lines 5, 6 of Alg.3, if the computational load u is less than the fixed (FTH_DOWN) 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. [10] and is detailed in Alg. 4. Differently from [10], 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 Alg. 3). As shown in Alg. 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 Alg. 5). If no neighbor in $otherNeighList$ can receive the vm, the $underNeighList$ is considered with a best-fit approach (lines 14-25 of Alg. 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 Alg. 5). $migrateAll(vmList)$ operates in a "all-or-none" way, such that the migrations are committed on the future maps (line 34 of Alg. 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 Alg. 5).

As shown in Fig. 5, 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* 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 tested it on DAM-Sim [11], a Java simulator able to apply a specific policy on a collection of neighborhoods through DAM protocol and to compare its performance with a centralized policy implementation.

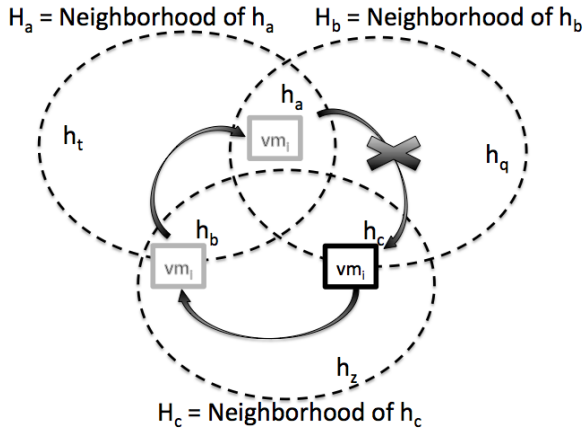


Fig. 5: Example of three overlapping neighborhoods.

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.

According to the tuning tests of Minimization of Migration algorithm [5], the FTH_DOWN and FTH_UP thresholds have been fixed at 25% and 95%, respectively, while we initially set the tolerance interval t for load balancing at 8%.

We 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. These hosts are randomly chosen in every experiment.

In Fig. 6, we compare the MWF performance with nN=5 and 10 nodes in each neighborhood, with the application of a centralized best fit policy (BF-GLO in Fig. 6a) [9]. We also show the performance of BF: a best fit policy applied in a distributed way by means of DAM protocol. BF exploits the two-phase lock protocol, therefore, each time a server detects to be underloaded or overloaded, it start reconsidering the current VM allocation for the whole neighborhood. Details about BF implementation can be found in [9].

Figs. 6a and 6d 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.

Figs. 6b and 6e 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:

$$nMig\frac{onServers}{nVM} \tag{2}$$

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 values 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.

Figs. 6c and 6f 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.

In Fig. 7, we can see the distribution of number of servers along load intervals. In the initial scenario (INI in Fig. 7) all the servers have 50% load except for 20 underloaded and 20 overloaded nodes.

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. 7, where almost all the switched-on servers are at 95% of utilization, 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 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. 7 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.

In order to provide a clearer idea of the efficacy of our approach, we separately tested MWF performances in terms of energy efficiency and load balancing. To this purpose, we built three different scenarios.

A. Scenario 1: load balancing test

Considering MWF from the load balancing perspective only, we created a collection of 50 initially unbalanced scenarios satisfying the constraint:

$$U_{TOT} > FTH\_UP(N-1) \tag{3}$$

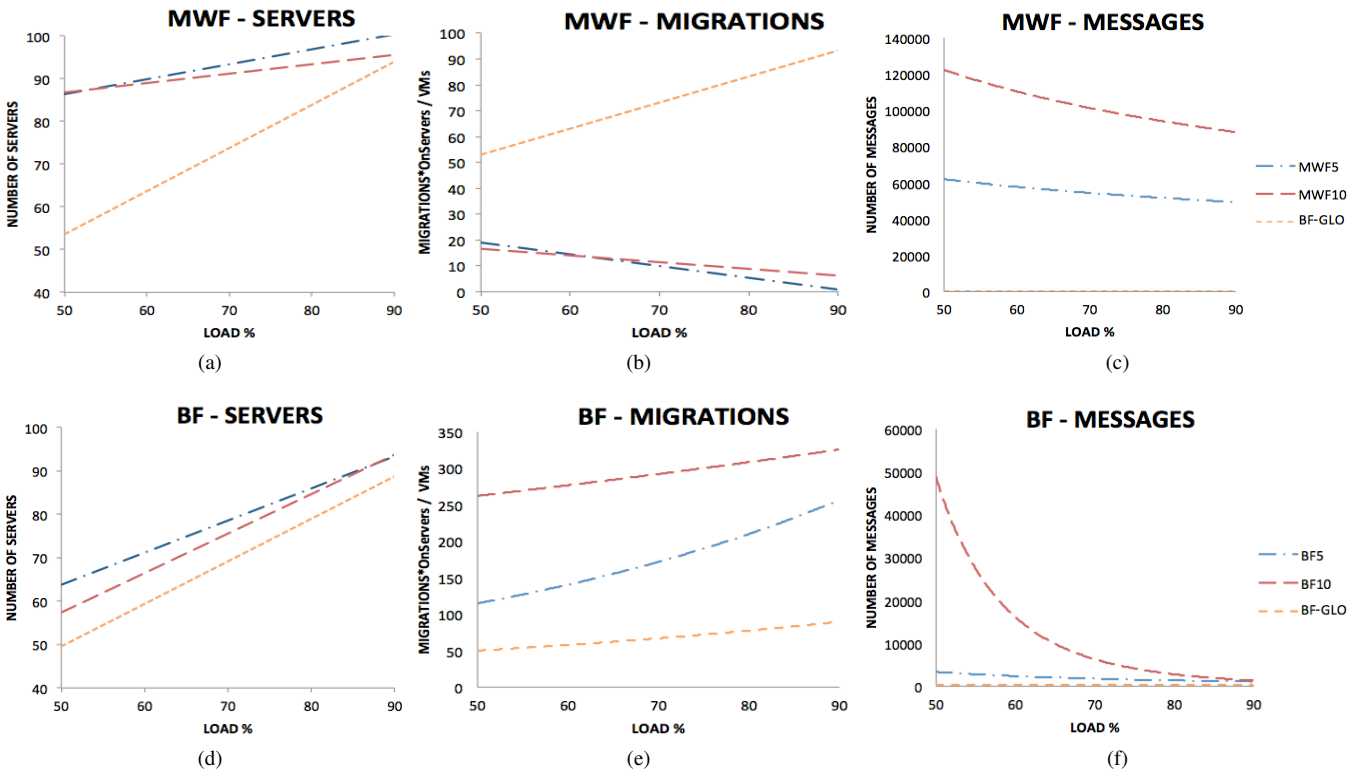


Fig. 6: MWF end BF performance comparison.

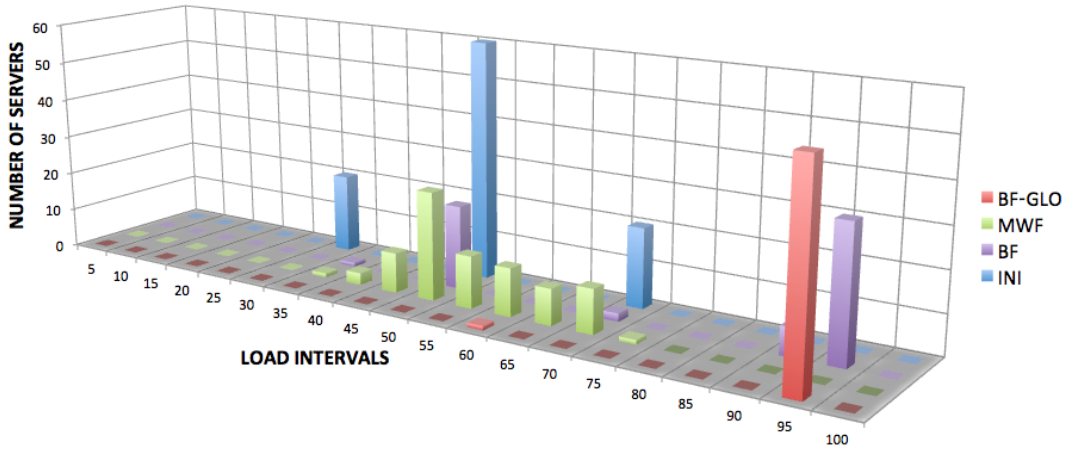


Fig. 7: Distribution of servers on load intervals.

where U_{TOT} is the total CPU-utilization of the datacenter and N is the number of simulated servers. In this way, we can ensure that no server switch-off is possible, and we can test the MWF load balancing performance only.

By defining $U_{AVG\_N} = U_{TOT}/N$ the average load over N servers, the relation 3 can be rewritten as follows:

$$U_{AVG\_N} > FTH\_UP\frac{N-1}{N} \quad (4)$$

and the initial scenarios can be built such that each server h has a CPU-utilization U_h uniformly distributed in the interval:

$$U_h \in [U_{AVG\_N} - q, U_{AVG\_N} + q] \quad (5)$$

where q expresses the degree of imbalance in the initial scenario. We tested the MWF performance with $FTH\_UP = 90\%$, $q = 10\%$ and averaged the results over 50 simulations.

In each scenario the topology of the neighborhoods is generated randomly.

Fig. 8 shows the distribution of servers over load intervals. In the initial scenario (INI) all the servers are on average loaded around the value of FTH_UP. We show the distribution after a global worst-fit optimization (WF-GLO in Fig. 8) and the application of MWF by means of DAM protocol with 10 as neighborhood size.

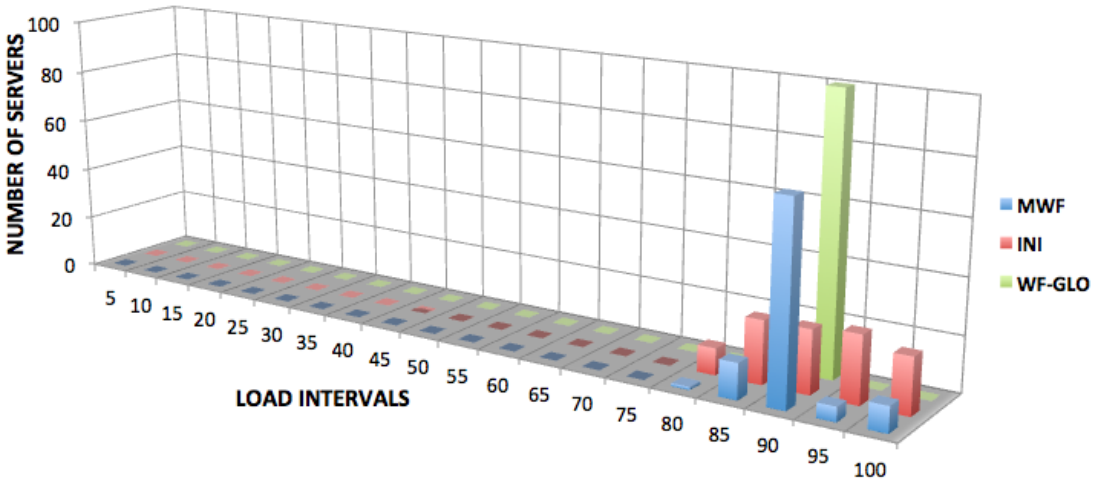


Fig. 8: Distribution of servers on load intervals.

MWF shows good performance from the load balancing perspective even if, as we expected, relying on a global knowledge of status of each server, the centralized application of a worst fit policy clearly outperforms the distributed approach.

B. Scenario 2: power saving test

In order to mainly test the energy saving performance of MWF, we create a collection of scenarios satisfying this constraint:

$$U_{TOT} = FTH\_UP \cdot M, s.t. M < N \tag{6}$$

where M is the number of servers that remains switched-on at the end of the optimization. Relation (6) can be rewritten as follows:

$$U_{AVG\_N} = FTH\_UP \cdot \frac{M}{N} \tag{7}$$

Therefore, given a certain U_{AVG_N} we can calculate the minimum number M_{opt} of servers that can execute the data-center's workload:

$$M_{opt} = \frac{U_{AVG\_N} \cdot M}{FTH\_UP} \tag{8}$$

We create a collection of scenarios with increasing values of U_{AVG_N}, having the load U_h of each server again uniformly distributed in the interval (5) and $q = 20\%$, and we use M_{opt} to evaluate the performance of MWF.

Figs. 9a, 9b and 9c show the number m of working servers at the end of different MWF distributed executions. These values are compared to the minimum possible number M_{opt} of running servers in each scenario.

Each point in the graphs of Fig. 9 represents an initial scenario with different value for U_{AVG_N}.

We repeated the experiment with three different values of the rate $^q/_t$. Fig. 9a shows the energy saving performance with $q = 15\%$ of imbalance in the initial scenario and $t = 5\%$ as MWF tolerance interval. The number of switched-off servers is far from the optimum value (expressed by the blu line) for every generated scenario, while decreasing the ratio $^q/_t$ to $^{20}/_5$ and $^{20}/_3$ (as reported by Figs. 9b and 9c) the performance of MWF significantly increases.

For low values of U_{AVG_N} the algorithm seems to perform significantly better for every value of the rate $^q/_t$, This effect is due to the FTH_DOWN, which is fixed at 25% in every scenario and can therefore contribute to make some $MasterClients$ start if the hosts are detected to be underloaded ($U_h < FTH\_DOWN$).

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.

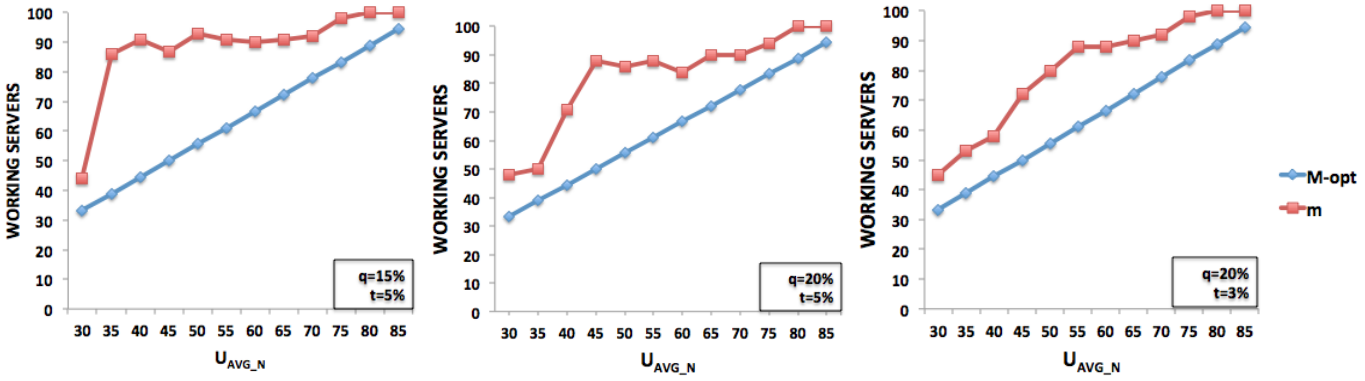
C. Scenario 3: scalability test

In order to test the scalability of the distributed approach, we analyzed MWF behavior while increasing the number of simulated servers and VMs up to 2000 and 60000, respectively. Fig. 10a shows the number of migrations stated by MWF and compare it with that of WF-GLO policy.

Since the number of VMs increases with the number of hosts, we actually compare the ratio between migrations and number of VMs in the scenario.

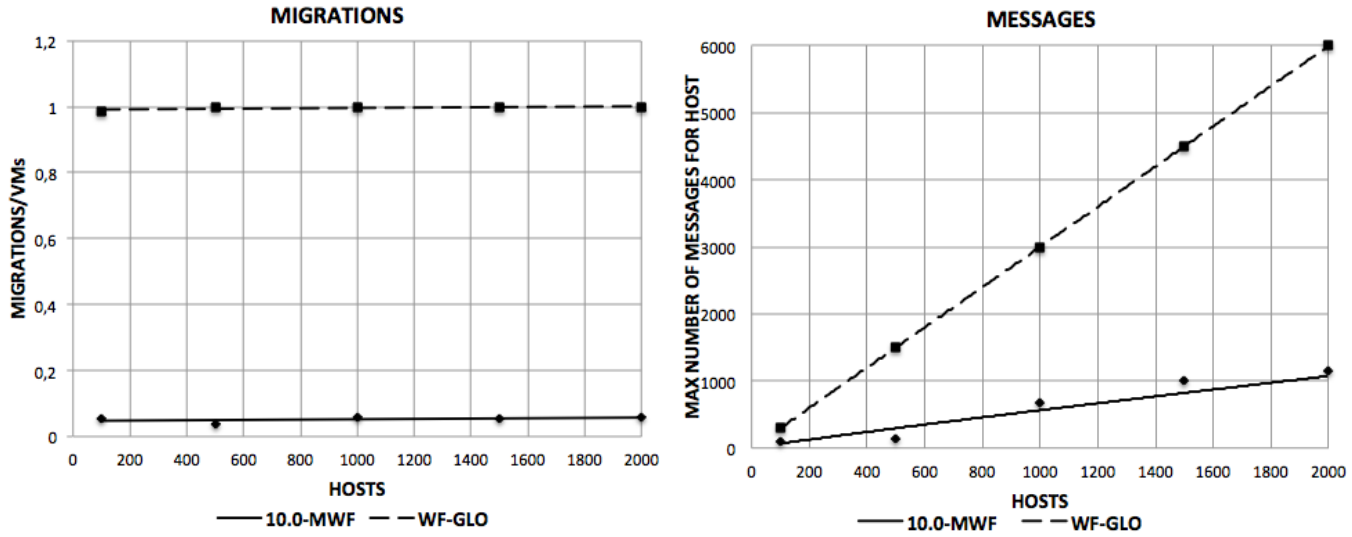
WF-GLO policies does not take into account the current allocation while performing the optimization, therefore, it results in a very high number of migrations, near to the total of VMs. Conversely, MWF distributed policy only operates on underloaded or overloaded nodes (with CPU utilization lower than FTH_DOWN or higher than FTH_UP, respectively) or on those hosts that are unbalanced in respect to the average of their neighborhood (CPU utilization out of the interval $[MTH\_DOWN, MTH\_UP]$). For this reason, as shown in Fig. 10a, the number of resulting migrations is significantly lower for MWF.

In Fig. 10b, we consider the maximum number of messages exchanged by a single host. Since WF-GLO is centralized, the coordinator node must collect the state of all the other nodes before starting the optimization and finally return the



(a) (b) (c)

Fig. 9: MWF power saving performance test. The number m of working servers at the end of different MWF executions is compared to the minimum possible number M_{opt} of running servers in each scenario. The experiments are repeated with different values of the ratio $^{q}/_{t}$.



(a) (b)

Fig. 10: MWF and WF-GLO performance comparison. 10a: Number of migrations performed for increasing number of simulated hosts. We compare the performance of MWF with tolerance interval 10.0 with centralized WF-GLO. 10b: Maximum number of messages exchanged (sent and received) by a single host of the datacenter. MWF significantly outperforms WF-GLO for high values of simulated hosts.

new configuration to each node. Therefore, as shown in Fig. 10b, the number of messages exchanged by the coordinator is always proportional to the number of the nodes it manages. The behavior of MWF policy is again proportional to the number of nodes but the trends is significantly lower. This comes from the fact that, according to MWF, each node of the datacenter always communicate with a predefined number of neighbors (5 in this simulation).

Therefore, considering the maximum number of messages exchanged by a node, for high values of simulated hosts, we can conclude that MWF distributed approach performs better than the centralized WF-GLO algorithm.

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 [3], [4], [12]. Some other solutions concern centralized energy-aware optimization algorithms [5], [13]–

[15], in particular extensions of the Bin Packing Problem [16], [17] to solve both VMs allocation and migration problems [10]. 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.

Focusing on the SLA compliance perspective only, some recent works have applied the known load balancing techniques (suitable for both distributed systems [18]–[20] and high performance computing [21], [22]) to the Cloud Computing paradigm, particularly as regards the VM allocation problem. A lot of works in this field focus on decentralized solutions [23]–[26] in order to obtain a higher rate of scalability and reliability. As pointed out by Randles et al. [27] comparative study, all these distributed policies again assume that each node can obtain a complete knowledge of the datacenter status. Conversely, our approach does not rely on this strong assumption because each node can work with a local view of the system status, limited by the size of the neighborhood.

Similarly to our approach, in the work by Zhao et al. [28], the decentralized load balancing policy relies not only on the static VM allocation, but also on live migrations in order to run-time dynamically relocate VMs. In [28], as in [27], each node of the datacenter must be able to access a global view of the current allocation scenario.

Furthermore, our work does not focus on the SLA compliance perspective only, but also considers VM consolidation strategies to obtain energy efficiency.

Finally, other approaches involve intelligent, optionally bio-inspired [29], [30], agent-based system, which can give to the datacenter a certain rate of independence from human administration, showing an intelligent self-organizing emergent behavior [31]–[33], and also provide the benefits of a more distributed system structure [34], [35].

As Mastroianni et al. [35] pointed out, building a distributed self-organizing and adaptive infrastructure for VM consolidation can lead to significant scalability performance improvement. As in [35], each physical node of our architecture is able to take decisions on the assignment and migration of VMs exclusively driven by local information. Yet, differently from [35], the assignment procedure of MWF algorithm does not involve all the servers in the datacenter, but only a fixed subset of neighbor nodes.

As in the work by Marzolla et al. [31], which is based on Gossip protocol [36], we adopt a self-organizing approach, where coordination of nodes in small overlapping neighborhoods leads to a global reallocation of VMs, but differently from [31] we created a more elaborate model of communication between physical hosts of the datacenter. In particular, while in [31] 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 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. Relying on DAM protocol, we investigated a VM migration approach (MWF) suitable for a distributed management in a cloud datacenter.

Evaluation of MWF policy by means of an ad hoc built software simulator 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 further investigations of performance on increasing size datacenters has shown that the decentralized nature of our approach can intrinsically contribute to augment the scalability of the cloud management infrastructure.

In the near future, we will extend DAM-Sim in order to take into account not only computational resources, but also memory and bandwidth requirements. This will allow us to test different and more elaborated reallocation policies. 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.

Further investigation will be necessary to address issues caused by message losses. We will enrich the algorithm with a recovery strategy in order to avoid the risk of physical servers never-ending blocked while they wait for ”unlock” messages.

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] D. Loreti and A. Ciampolini, “Policy for distributed self-organizing infrastructure management in cloud datacenters,” in ICAS 2014, The Tenth International Conference on Autonomic and Autonomous Systems, IARIA, Ed., 2014, pp. 37–43.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] A. Marinos and G. Briscoe, "Community cloud computing," in First International Conference, CloudCom 2009. Proceedings. Springer Berlin Heidelberg, 2009, pp. 472–484.

[7] C. Giovanoli and S. G. Grivas, "Community clouds a centralized approach," in CLOUD COMPUTING 2013, The Fourth International Conference on Cloud Computing, GRIDs, and Virtualization, IARIA, Ed., 2013, pp. 43–48.

[8] K. Chard, S. Caton, O. Rana, and K. Bubendorfer, "Social cloud: Cloud computing in social networks," in Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, July 2010.

[9] 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 [Accesed 20th November 2014].

[10] 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.

[11] [Online]. Available: https://bitbucket.org/dloreti/cooperating.cloud.man [Accesed 20th November 2014]

[12] 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.

[13] 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.

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

[15] J. C. adn Weidong Liu and J. Song, "Network performance-aware virtual machine migration in data centers," in CLOUD COMPUTING 2012 : The Third International Conference on Cloud Computing, GRIDs, and Virtualization, IARIA, Ed., 2012, pp. 65–71.

[16] 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.

[17] 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.

[18] A. Piotrowski and S. Dandamudi, "A comparative study of load sharing on networks of workstations," in Int. Conf. on Parallel and Distributed Computing Systems, 1997, pp. 458–465.

[19] A. N. Tantawi and D. Towsley, "Optimal static load balancing in distributed computer systems," Journal of the ACM, vol. 32, no. 2, pp. 445–465, April 1985.

[20] T. Chou and J. Abraham, "Load balancing in distributed systems," IEEE Transactions on Software Engineering, vol. 8, pp. 401–412, July 1982.

[21] G. Aggarwal, R. Motwani, and A. Zhu, "The load rebalancing problem," Journal of Algorithms, vol. 60, no. 1, pp. 42–59, July 2006.

[22] B. J. Overeinder, P. M. A. Sloot, R. N. Heederik, and L. O. Hertzberger, "A dynamic load balancing system for parallel cluster computing," in Future Generation Computer Systems, vol. 12, 1996, pp. 101–115.

[23] F. Saffre, R. Tateson, J. Halloy, M. Shackleton, and J. L. Deneubourg, "Aggregation dynamics in overlay networks and their implications for self-organized distributed applications," The Computer Journal, vol. 52, no. 4, pp. 397–412, 2009.

[24] O. A. Rahmeh, P. Johnson, and A. Taleb-bendiab, "A dynamic biased random sampling scheme for scalable and reliable grid networks," INFOCOMP - Journal of Computer Science, vol. 7, no. 4, pp. 1–10, December 2008.

[25] S. Nakrani, C. Tovey, S. Nakrani, and C. Tovey, "On honey bees and dynamic server allocation in internet hosting centers," Adaptive Behavior, vol. 12, pp. 223–240, 2004.

[26] T. Suzuki, T. Iijima, I. Shimokawa, T. Tarui, T. Baba, Y. Kasugai, and A. Takase, "A large-scale power-saving cloud system with a distributed-management scheme," in International Journal on Advances in Intelligent Systems, vol. 7. IARIA, 2014, pp. 326–336.

[27] M. Randles, D. Lamb, and A. Taleb-Bendiab, "A comparative study into distributed load balancing algorithms for cloud computing," in Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on, April 2010, pp. 551–556.

[28] Y. Zhao and W. Huang, "Adaptive distributed load balancing algorithm based on live migration of virtual machines in cloud," in INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on. IEEE, 2009, pp. 170–175.

[29] 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.

[30] 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.

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

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

[33] A. Esnault, "Energy-aware distributed ant colony based virtual machine consolidation in iaas clouds," Master's thesis, Universitè de Rennes, 2012.

[34] 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.

[35] C. Mastroianni, M. Meo, and G. Papuzzo, "Probabilistic consolidation of virtual machines in self-organizing cloud data centers," IEEE Transactions on Cloud Computing, vol. 1, no. 2, pp. 215–228, 2013.

[36] 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.