Two levels autonomic resource management in virtualized IaaS

Alain Tchana, Giang Son Tran, Laurent Broto, Noël de Palma, Daniel Hagimont

Two levels autonomic resource management in virtualized IaaS

Alain Tchana [a,*], Giang Son Tran [b], Laurent Broto [b], Noel DePalma [a], Daniel Hagimont [b]

[a] University of Joseph Fourier, (UJF/LIG) - 621, avenue Centrale SAINT-MARTIN-D'HERES, BP 53, 38041 Grenoble cedex 9, France
[b] University of Toulouse, (IRIT/ENSEEIHT) - 2, rue Charles CAMICHEL B.P. 7122 31071 TOULOUSE Cedex 7, France

A B S T R A C T

Virtualized cloud infrastructures are very popular as they allow resource mutualization and therefore cost reduction. For cloud providers, minimizing the number of used resources is one of the main services that such environments must ensure. Cloud customers are also concerned with the minimization of used resources in the cloud since they want to reduce their invoice. Thus, resource management in the cloud should be considered by the cloud provider at the virtualization level and by the cloud customers at the application level. Many research works investigate resource management strategies in these two levels. Most of them study virtual machine consolidation (according to the virtualized infrastructure utilization rate) at the virtualized level and dynamic application sizing (according to its workload) at the application level. However, these strategies are studied separately. In this article, we show that virtual machine consolidation and dynamic application sizing are complementary. We show the efficiency of the combination of these two strategies, in reducing resource usage and keeping an application's Quality of Service. Our demonstration is done by comparing the evaluation of three resource management strategies (implemented at the virtualization level only, at the application level only, or complementary at both levels) in a private cloud infrastructure, hosting typical JEE web applications (evaluated with the RUBiS benchmark).

1. Introduction

In order to reduce the maintenance cost of computing environments, companies are increasingly externalizing their computing infrastructures to specific companies called *providers*. These later are expected to ensure quality of service (QoS) for their *customers* while minimizing hosting costs. Cloud computing is a recent paradigm which follows this direction.

In this context, on demand resource management is one of the main services that such an environment must ensure. It must allow the allocation of resource as needed and resource deallocation when unused, while reducing the number of used machines and therefore energy consumption. Resource usage reduction also concerns cloud customers since they want an efficient service while using the minimum number of resources in the cloud (which impacts their invoice).

Many research and industry works ([1,2], or [3]) have investigated resource management strategies in the cloud. These strategies can be grouped into two categories: those which are implemented at the customer applications level through dynamic application sizing (according to its workload) and others that are implemented at the virtualized level through virtual machine consolidation (according to the cloud infrastructure utilization rate). However, (1) these strategies have been studied and experimented separately, and (2) they do not consider each level particularity regarding resource management.

In this article, we investigate the implementation of resource management strategies at both levels simultaneously in the case of an IaaS (Infrastructure-as-a-Service) cloud model hosting master–slave applications. We show that virtual machine consolidation and dynamic application sizing can be beneficently combined in order to reduce resource usage and to keep application's Quality of Service (QoS). Our study is done by comparing the evaluation of three resource management strategies (implemented at the virtualization level only, at the application level only, and complementary at both levels) in a private IaaS, hosting typical JEE web applications (evaluated with the RUBiS [4] benchmark). Our private IaaS cloud infrastructure automates the execution of these strategies with an autonomic management system called TUNe [5].

The rest of this paper is organized as follows. Section 2 presents the context of this work. Section 3 presents our motivations for two-level resource management in a cloud infrastructure. Section 4 presents the environment in which we conducted our experiments. Sections 5–8 detail, evaluate and synthesize the resource

* Corresponding author. Tel.: +33 620740733.
E-mail addresses: alain.tchana@inria.fr (A. Tchana), giang.tran@enseeiht.fr (G. Son Tran), laurent.broto@enseeiht.fr (L. Broto), noel.de_palma@inria.fr (N. DePalma), daniel.hagimont@enseeiht.fr (D. Hagimont).

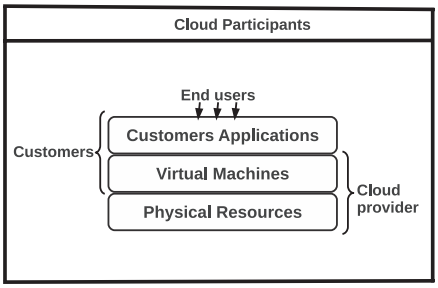

Fig. 1. Cloud computing layers and participants.

management strategies we study in this article. Section 9 presents the various related works. Finally we conclude and present future works in Section 10.

2. Context

In this section, we introduce the context of cloud environments in which our experiments take place and the type of applications we are considering.

2.1. Cloud computing

Using a cloud computing environment, an organization (*customer*) can greatly reduce maintenance costs by relying on an external institution, called *provider*. Resources in a cloud computing environment are dynamically provisioned and must satisfy service-level agreements between the *provider* and its *customers* [6].

In the literature, cloud infrastructures are generally classified into three models:

- Infrastructure-as-a-Service (IaaS): a virtualized infrastructure managed by a provider, in which external customers deploy and execute their applications;
- and Platform-as-a-Service (PaaS): a virtualized infrastructure managed by a provider, in which customers develop, deploy and execute their applications using the application development environment provided by the provider;
- Software-as-a-Service (SaaS): the provider gives to customers the application they want.

In this paper, we consider the cloud as an IaaS. The actors involved in such a cloud platform are grouped into three categories: cloud providers, cloud customers and end users. A *cloud provider* is responsible for the administration of the cloud resources (e.g. hardware, network, virtual machine) and services. *Cloud customers* use the provided cloud resources to deploy and execute their applications (e.g. JEE applications in our example). Cloud customers are provided with VMs, without having a global view or direct access to the cloud physical infrastructure. These VMs host the customer's applications and represent a confined portion of physical resource. *End users* use customer applications deployed in the cloud.

Fig. 1 summarizes these cloud layers and the scope of each cloud participant: the cloud provider has access to physical resources and VMs; cloud customers have access to VMs and their applications; and end users have access to the customer applications.

2.2. Master–slave applications: example of JEE applications

Typical web applications in Java 2 Platform Enterprise Edition (JEE) are designed as master–slave architectures. This type of application represents the commonly hosted applications in cloud platforms. Its design consists of a web server tier (e.g. Apache), an application server tier (e.g. Tomcat) and a database server tier (e.g. MySQL). When an HTTP request is received, it refers either to a static web document (e.g. HTML, CSS), in which case the web server directly returns the requested document to the client; or to a dynamically generated document, in which case the web server forwards the request to the application server. In turn, the application server executes requested application components (e.g. Servlets, EJBs), creating queries to a database through a JDBC driver (Java DataBase Connection driver). Finally, queried data from the database is processed by the application server to generate a web document which is returned to the client. Fig. 2 summarizes this architecture of an JEE application.

In this context, the increasing number of Internet users has led to the need for highly available services. To face high loads of Internet services, a commonly used approach is the replication of servers. This kind of approach usually defines a particular software component in front of each set of replicated servers, which dynamically balances the load among the replicas using different algorithms (e.g. Random choice, Round Robin). Server replication is a way to implement elastic configuration through dynamic sizing of the number of replicas.

Here, an important remark is that these JEE applications are mostly CPU bound, i.e. the goal of dynamic sizing is the balance the CPU load between a variable number of replicas.

3. Motivations

As any business activity, the main objective of a cloud provider is to make a profit while meeting customers requirements (which essentially means to guarantee that all their VMs will receive the contracted resources). This objective can be achieved in two ways.

Firstly, the provider can strictly keep allocated all resources the customer has contracted (for the entire VM lifetime). Consequently, each VM will keep its resources even if it does nothing.

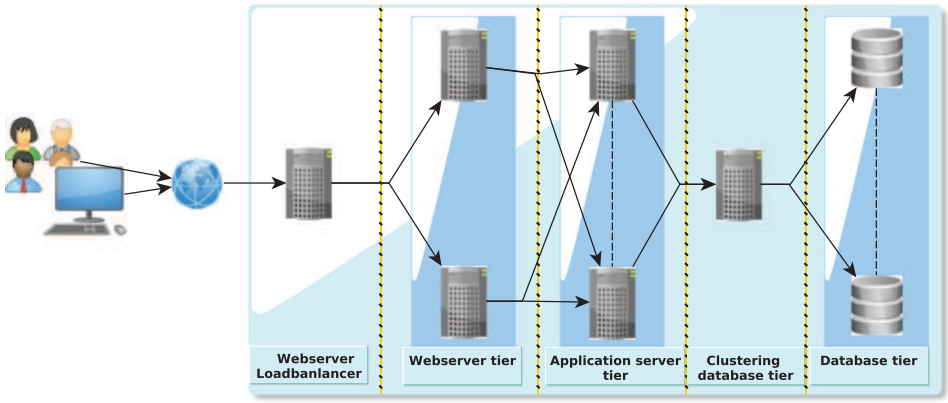

Fig. 2. A JEE application architecture.

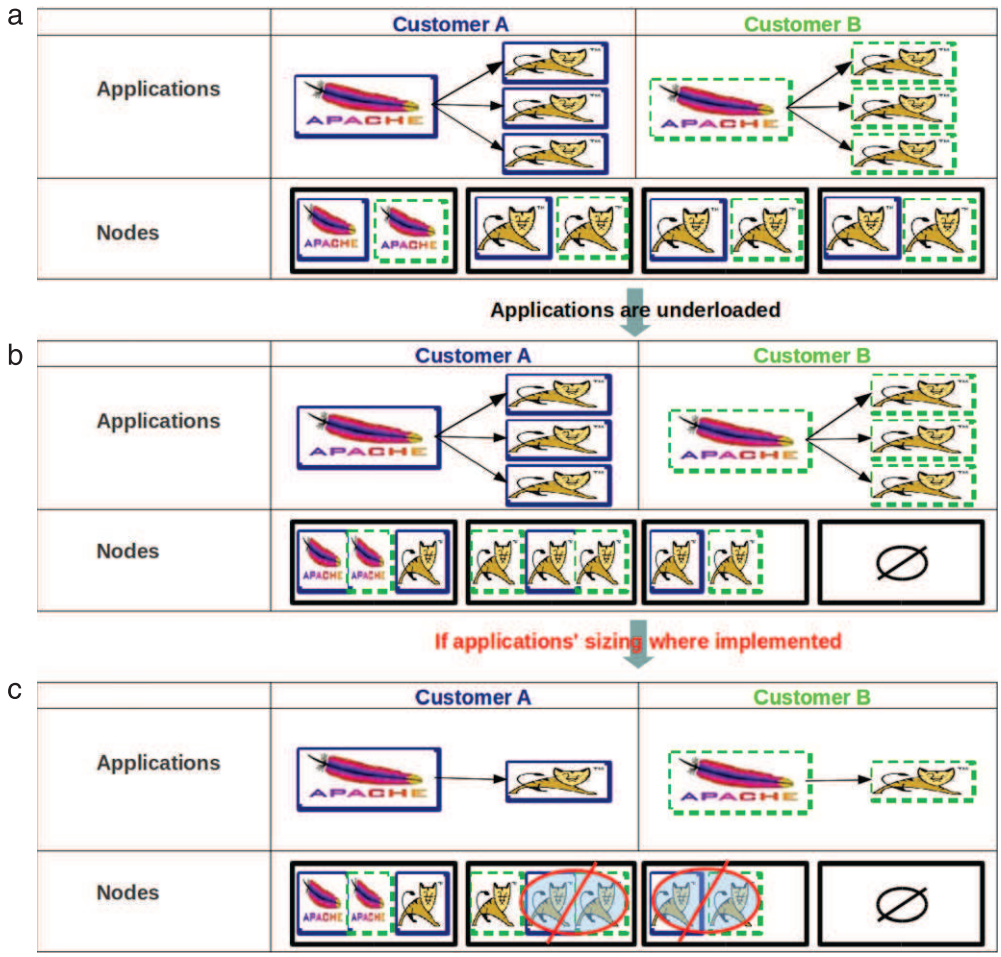

Fig. 3. Static application configuration—VM consolidation.

Since this way reduces the cloud mutualization capacity (it is not possible to group idle VMs on fewer machines because their resources are booked), VM allocation in the cloud will be expensive.

In the second way, the provider ensures to customers that their VMs will received all resources they need (in the limit of the contract) only when the need is effective. Thus, the provider can increase his hosting capacity by collocating VMs (using VM migration) when they are underused, and relocating (spreading) them when they are effectively used. Consequently, VM allocation can be cheaper since the provider makes profit by increasing his hosting capacity (which implies hosting more customer applications). VM collocation and relocation are known as *VM consolidation* strategies.

In short, cloud providers can provide two VM reservation modes:

- *Hard VM* reservation. Resources are kept allocated.
- *Soft VM* reservation. Resources are only allocated when effectively used.

At the application level, customers can either implement a static or a dynamic configuration strategy.

With a static configuration strategy, a customer will start the number of server replicas needed to face expected peak loads in order to ensure application QoS. Each server replica is created in a separate VM. Then, the allocated VMs are not always intensively active and the customer can therefore rely on Soft VMs. The IaaS consolidation system will dynamically allocate resource to these VMs according to their load.

With a dynamic configuration strategy, a customer monitors VMs load and adapts the degree of replication according to this load, adding a new replica (on a new VM) when the application workload exceeds a threshold; and removing a replica when the load decreases. In this case, this dynamic policy ensures that VMs are effectively used and the customer can therefore rely on Hard VMs.

In summary, two resources management strategies can be implemented in a cloud running master–slave applications:

- *Static application configuration with Soft VMs*.
- *Dynamic application configuration with Hard VMs*.

These two strategies are analyzed below.

3.1. Static application configuration with Soft VMs

Let us consider a situation where the cloud runs two multi-tier applications (an Apache web server linked to one or many Tomcat application servers), owned by different customers. Each tier is run on a separate VM which corresponds to a fraction of a physical machine. Applications are statically dimensioned with the maximum number of replicas needed to maintain their QoS under peak loads: three Tomcat instances for each application are initially allocated to serve peak loads. Fig. 3(a) shows an overloaded state of the applications. As applications become underloaded, VM consolidation progressively gathers VMs on a minimal number of physical machines (Fig. 3(b)). However, as argued in [7], an important limitation of VM consolidation is memory capacity: any VM, even idle, consumes physical memory, which limits the number of VMs that can be hosted on a machine. Therefore, VM

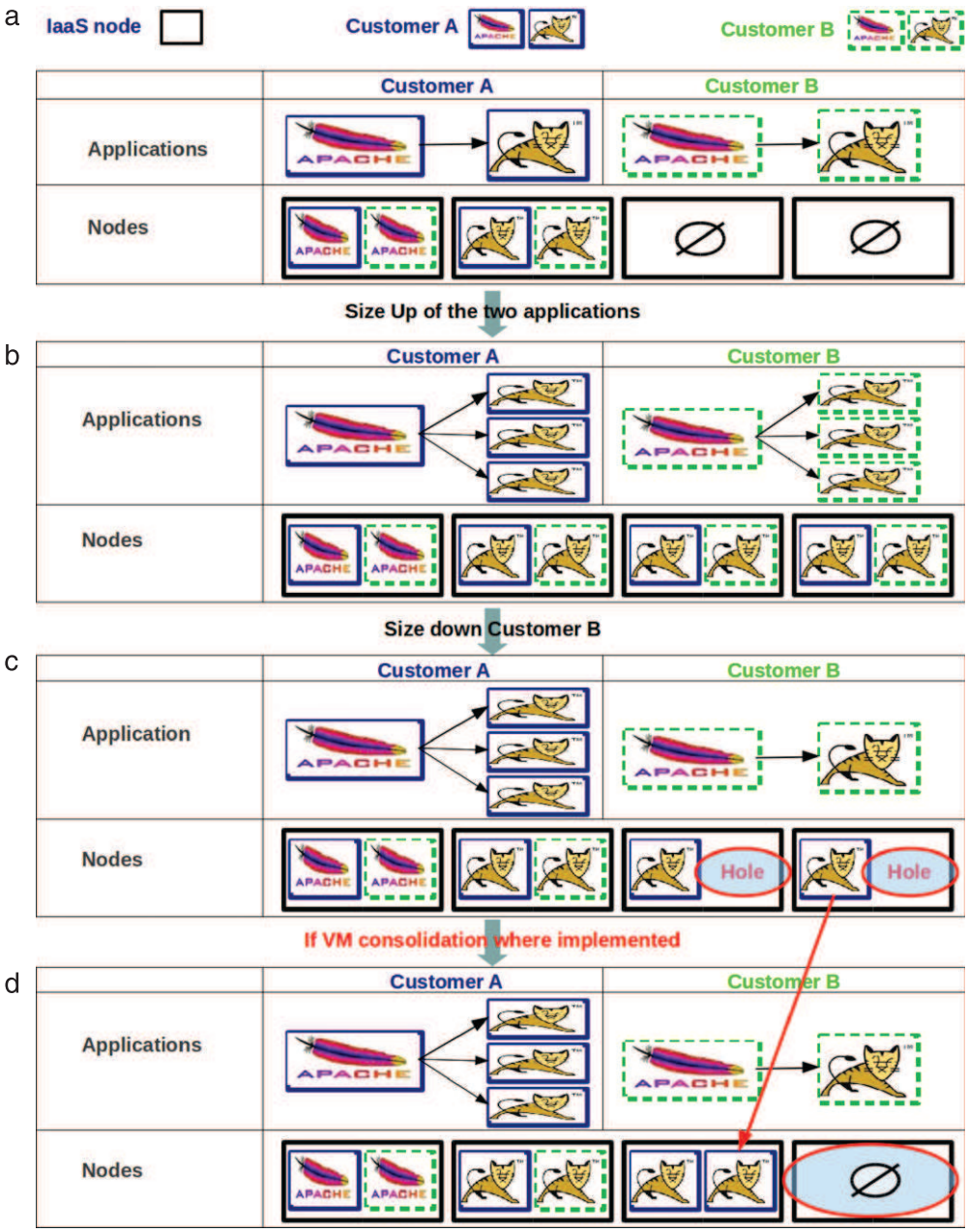

Fig. 4. Dynamic application configuration.

consolidation can reduce, but cannot minimize the number of active machines in the hosting center.

Dynamic application configuration (sizing) would allow to significantly reduce the number of physical machines in use (Fig. 3(c)).

3.2. Dynamic application configuration with Hard VMs

We consider an initial situation where each application is configured with one Tomcat server (Fig. 4(a)). The customers execute two sizing operations: size up of the two applications (each has three Tomcats) facing a peak load (Fig. 4(b)), followed by a size down of the second application when it becomes underloaded (Fig. 4(c)). This last operation leaves two unused spaces in the last two physical machines. These holes can be easily optimized if VM consolidation is implemented (Fig. 4(d)), resulting in a physical machine being freed for energy saving.

Dynamic application sizing ensures that VM resources are effectively used (else VM would be deallocated), so that this solution can rely on Hard VM allocation. Since VMs are effectively used, the VM consolidation algorithm does not have to monitor CPU. It simply packs VMs on the minimal number of physical machines according to the resource allocated to these Hard VMs. We observe here that this solution combines both application sizing and VM consolidation, but the consolidation system is in this case much simpler.

This analysis shows the theoretical advantage of two-level resource management. In the following, we describe an implementation of the above strategies and an evaluation in a private cloud infrastructure.

4. Evaluation context

This section describes the IaaS environment used to evaluate the resource management strategies presented above. Our experimental context is a private virtualized infrastructure, equipped with an autonomic administration system (called TUNe) which provides self-* capabilities.

4.1. TUNe: an adaptable autonomic administration system

The TUNe [5] system aims at bringing a solution to the increasing complexity in distributed software management. It is based on the concept of autonomic computing. An autonomic management system automates the management of applications without human intervention. It covers applications' life cycle: deployment, configuration, launching, and dynamic management. TUNe allows the definition of managers which monitor the execution conditions and react to events such as failures or peak loads, in order to adapt the managed application accordingly and autonomously.

TUNe has been experimented in various application domains: web and middleware architectures [8], grid computing systems [9] and energy saving in replicated systems [10].

This adaptability to different domains was made possible thanks to the component-based approach used to design TUNe. Each legacy software is wrapped inside a software component, and is managed by its component's management interface. TUNe supports the introduction of new description languages to define software and hardware architectures as well as system management policies. For example, TUNe's deployment policies are highly adaptable, from binary file transfer to NFS remote directory mounting. The definition of reconfiguration policies is also flexible, allowing to implement autonomic management behaviors for application software as well as operating systems (virtual machines). In summary, TUNe is designed and developed with pluggable components, even administrative services are implemented as components. Thus, in the context of this article, we were able to adapt TUNe to manage both customer applications (with dynamic sizing) and the virtualized cloud environments (with consolidation).

4.2. Experimental JEE application

The RUBiS [4] benchmark (version 1.4.2) is used as our customer application throughout this article. RUBiS is an implementation of an eBay-like auction system, including a workload generator to simulate web clients. RUBiS is an implementation of a JEE benchmark. In our experiments, our RUBiS application is composed of an Apache web server, a Tomcat servlet container, a MySQL-Proxy to load balance SQL requests among a set of MySQL database servers. We limit our evaluation to the MySQL tier due to paper length restriction. Moreover, as identified by [11], the database tier is the primary bottleneck tier in such applications. The workload submitted to the RUBiS applications is database bound and our sizing policies are applied only on the MySQL tier. However, we treated in a previous work [12] the dynamic sizing of each JEE tier. For these experiments, we consider that our IaaS infrastructure hosts two RUBiS applications which belong to different customers.

4.3. Experimental environment

Our experiments were carried out using the Grid'5000 [13][1] experimental testbed (the French national grid), using Xen [14] 3.2 as the virtualization platform. All nodes use Linux CentOS distribution, running on a Dell PowerEdge R410 Intel Xeon E5620 2.4 GHz, and are connected through a gigabyte Ethernet LAN connection from a cluster. VM disk images shared by all the nodes are located on a NFS server. To summarize, the use of TUNe running over a Grid'5000's cluster is considered as our IaaS. The number of nodes during these experiments varies according to the dynamic changes of the workload submitted to the RUBiS application. For the two experimented applications, we use 3 nodes to run VMs (a total of 6 VMs) which host frontend servers (Apache, Tomcat and MySQL-Proxy), and up to 3 nodes (when the two applications receive their maximum load) to run VMs which host backend servers (MySQL, up to 6 VMs).

[1] Grid'5000 is an initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners.

4.4. Experimental procedure

The goal of our experiments is to demonstrate the benefit of combining an application level policy (dynamic sizing) and an IaaS level policy (consolidation). These experiments are done in three phases.

- In a first step, we execute a scenario (two RUBiS applications with a given workload) with static application configurations and Soft VMs. This means that here, resources are only managed by the consolidation system in the IaaS.
- In a second step, we execute the same scenario with dynamic application configurations and Hard VMs. This means that here, the degree of replication of application tiers is adapted according to the received load. In this step, we observe that holes may appear in the IaaS (as depicted in Section 3.2) and that a form of consolidation is required at the IaaS level.
- In a last step, we add the required consolidation system to the previous experiment, thus combining the two resource management policies (dynamic sizing at the application level and consolidation at the IaaS level).

5. Resource management at virtualized level only

5.1. Resource management policy

In this case, applications are deployed in a static configuration (without runtime reconfiguration) with the maximum number of replicas to prevent peak loads. All the replicas are deployed on Soft VMs which are created on a minimal number of physical nodes. A resource management policy is only implemented at the IaaS level. The IaaS administrator specifies the consolidation policy based on sensors (for VMs and physical nodes) as follows:

- If a physical machine saturates, a VM is migrated to another physical machine with enough resources to accept it. If necessary, a new physical machine is switched on. We call this *VM relocation*.
- If a physical machine is underloaded and its hosted VMs can be accepted by other physical machines, these VMs are migrated. This physical machine can be freed and switched off. We call this *VM collocation*.

Therefore, VM consolidation implies the implementation of two algorithms: VM relocation (when overloaded) and VM collocation (when underloaded). Designing an efficient VM consolidation policy has to take into account live migration cost. Even if several research efforts are made to minimize migration cost [14], the multiplication of this operation in a short time significantly affects the execution of applications.

Algorithm 1 presents the VM relocation algorithm (when the CPU load of a machine is over a predefined maximum CPU threshold). This algorithm is summarized as follows. Firstly, it identifies the most loaded node. Then it evaluates for each VM on this node the slope of its CPU variation (VM CPU load are logged over a period). These slopes allow us to identify the VM responsible for the node overload. The principle of our VM relocation algorithm consists in giving CPU power to the overloaded VM as much as possible. This objective can be reached either by relocating this

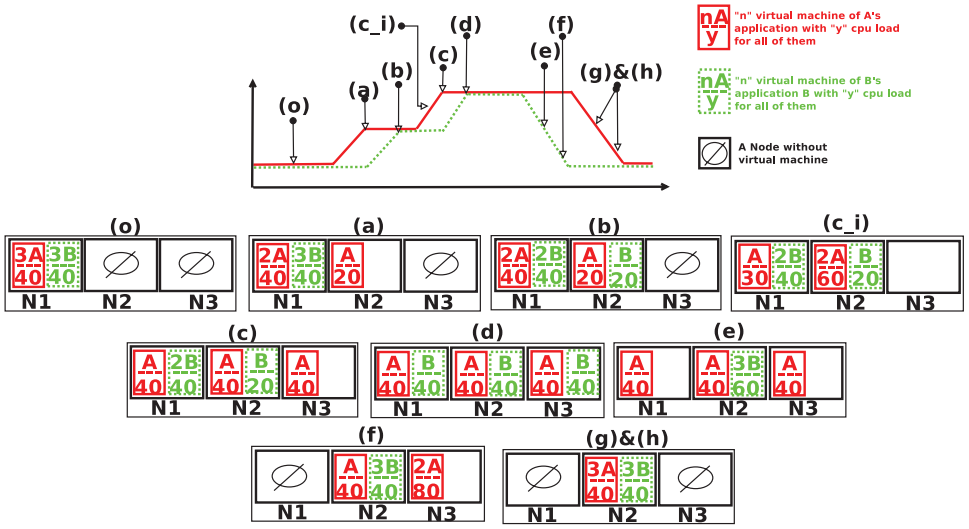

Fig. 5. Management at *IaaS level* only: Load scenario and VM placement status.

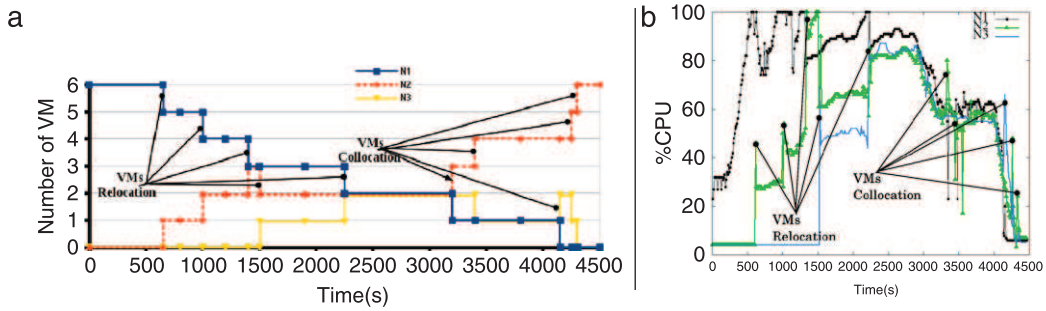

Fig. 6. Management at *IaaS level* only: (a) VM allocation on nodes and (b) node loads.

VM or by relocating another VM (so that the overloaded VM will stay on the initial node but with more CPU); we choose the relocation which gives the maximum CPU to the overloaded machine (without allocating a new physical machine). If no such relocation is found, a new machine is allocated and the VM is relocated to this machine.

Regarding VM collocation (presented in Algorithm 2), for each VM on the most underloaded node, we look for the Best-Fit node which can welcome this VM without reaching its maximum CPU threshold. If this was the last VM on the left node, the machine can be freed.

5.2. Evaluation

At the application level, two customers' applications are statically dimensioned and deployed with the maximum number of MySQL server (three per application). This management of the IaaS is ensured by the TUNe autonomic system. At creation time, TUNe maximizes the number of VMs per physical machine. This policy explains the fact that all VMs are hosted on node N1 (six VMs, Fig. 5) at the beginning of our experiment. A *Scheduler* component (implemented in TUNe) periodically receives and processes all VM and physical machine loads from remote probes and migrates VMs when necessary. It implements our consolidation policy presented in the previous subsection.

Fig. 5 shows the generated workload submitted to the two RUBiS applications and the IaaS node allocation status during the experiment. The representation "3A/40" at situation (o) means that this node hosts three VMs belonging to application A and the current CPU load of all three VMs is 40% of the capacity of the processor. Fig. 6 presents the results we obtained regarding the evolution of the number of VMs per node and the nodes' CPU loads (N1–N3). We discuss these results and compare them with those predicted in Fig. 5:

- Time (o): all VMs are hosted on N1.
- Time (a): the load increase for application A caused a migration of a VMA (VM of application A) on a new node (N2).
- Time (b): the load increase for application B caused a migration of a VMB, since it gives more capacity to the saturating VMB (on N2) than if we had migrated another VM from N1.
- Time (c_i): the load increase for application A caused a migration of a VMA from N1 to N2 (migrating a VMB from N1 would not do better).
- Time (c): the load increase for application A caused a migration of a VMA from N2 to N3 (migrating a VMB from N2 to N1 would saturate N1).
- Time (d): the load increase for application B caused a migration of a VMB from N1 to N3.
- Time (e): the load decrease for application B caused migrations of VMBs to N2.
- Time (f): a VMA is migrated from N1 to N3 (the most loaded node).
- Times (g and h): two VMAs are migrated to N2.

Algorithm 1 VMs Relocation (Virtualized Level Only)

Symbols:

- *MaxThreshold*: *The maximum acceptable CPU load on a IaaS node*
- *CurrentCPULoadOn*(N_i): *The current CPU load on node* N_i
- *epsilon*: *VM live migration overhead (in term of CPU)*#The migration process requires CPU on the #destination node.
- *NbOfVMsOn*(N_i): *A function which returns the number of VMs running on node* N_i

Begin

$N_o \leftarrow$ *The Most Loaded Node*
$VM_o \leftarrow$ *The VM with the greatest CPU slope on* N_o
# The objective of this algorithm is to give move CPU to VM_o.
# *How much CPU will remain on a node* N_i *if* VM_o *is migrated from* N_o *to* N_i?
for Each *IaaS's node* N_i *with* $NbOfVMsOn(N_i) > 0$ *and* $N_i \neq N_o$ **do**
  $AvailableCPUOn(N_i) \leftarrow MaxThreshold - CurrentCPULoadOn(N_i)$# Available CPU on N_i.
  $RemainCPUOn(N_i, VM_o) \leftarrow -1$# *RemainCPUOn*($N_i$, V_i): *CPU given to* VM_o *if* V_i *is migrated to* N_i.
  if $AvailableCPUOn(N_i) > CPU(VM_o) + epsilon$ **then**
    $RemainCPUOn(N_i, VM_o) \leftarrow AvailableCPUOn(N_i) - CPU(VM_o)$
  end if
end for
# *How much CPU will remain on* N_o *if another VM is migrated to another node*?
for Each VM_i *on* N_o *and* $VM_i \neq VM_o$ **do**
  for Each *IaaS's node* N_j *with* $NbOfVMsOn(N_j) > 0$ *and* $N_j \neq N_o$ **do**
    $RemainCPUOn(N_j, VM_i) \leftarrow -1$
    if $AvailableCPUOn(N_j) > CPU(VM_i) + epsilon$ **then**
      $RemainCPUOn(N_j, VM_i) \leftarrow AvailableCPUOn(N_o) + CPU(VM_i)$
    end if
  end for
end for
# We try to relocate either VM_o or another VM.
# The destination node can be a new IaaS node (which was not previously on).
(*MigratedVM*, *DestinationNode*) $\leftarrow$ VM_i *and* N_i *so that* $RemainCPUOn(N_i, VM_i)$ *is the highest*
if $RemainCPUOn(DestinationNode, MigratedVM) = -1$ **then**
  DestinationNode $\leftarrow$ *A new machine*
  DestinationNode $\leftarrow VM_0$
end if
Migrate MigratedVM from N_o *to DestinationNode*

End

All CPU peaks on Fig. 6(b) correspond to VM migrations and are due to live migration costs. Since our collocation algorithm considers that all nodes are identical in terms of hardware configuration, our experiment ends with all VMs on N2 instead of N1 (as at startup).

Despite the benefits in terms of the number of used machines, VM consolidation has important limitations when implemented without dynamic sizing. First, the number of VMs collocated on the same node is limited by node size (in terms of memory). In our experiment, a node can support up to six VMs at the same time. Secondly, VM overhead increases with the number of VMs on the node. We evaluated this overhead in a previous work [8]. Finally, there are many situations where several VMs hosting replicas of the same application tier (e.g. MySQL for application A) were hosted on the same node. For example, at time (a), node N1 hosts 2 MySQL instances of application A and 3 instances of application B. Dynamic sizing would avoid such situations.

6. Resource management at the application-level

6.1. Resource management policy

Dynamic sizing at application level (also known as dynamic server provisioning) consists of adding or removing replicas according to the monitored load of each tier. In that way, new resources are requested only when necessary, while ensuring the application QoS. Each replica is deployed and launched on a separate VM. The customer chooses an initial number of replicas for each tier and defines a dynamic sizing policy which generally takes the form of a threshold that a monitored load should not exceed. Sensors monitor each tier and generate an event whenever a constraint is violated. The reaction to an *overload* event is presented in Algorithm 3. On the other side, the reaction of an *underload* event is presented in Algorithm 4. As we previously mentioned, this policy only makes sense when applications allocate Hard VMs. Obviously, it may leave holes (unused resource portions) on IaaS machines if no consolidation mechanism is implemented at this level, as illustrated in the next section.

6.2. Evaluation

The objective of this evaluation is twofold: (1) to confirm the results of many other research works that show the benefit of dynamic sizing; and (2) to show the limits of this policy as identified in Section 3.

We based our self sizing policy on the average CPU load of MySQL servers. TUNe is used at the application level to implement this policy. Therefore, each MySQL VM is equipped with a monitoring agent, informing its TUNe manager about the variation of the

Algorithm 2 VMs Co-location (Virtualized Level Only)

Symbols:

- This algorithm uses the same symbols as Algorithm 1

Begin

$N_u \leftarrow$ *The most underloaded node*# The most underloaded node.
We try to free the most underloaded node N_u.
All its VMs will be relocated (if possible) on the other underloaded nodes.
$AvailableCPUOn(N_u) \leftarrow MaxThreshold - CurrentCPULoadOn(N_u)$
for Each VM_i *on* N_u, *sorted in a decreasing order* **do**
  $Rest_{min} \leftarrow 100$
  # What is the Best-Fit node for the relocation of VM_i?
  $DestinationNode(VM_i) \leftarrow NULL$
  for Each *IaaS's node* N_j *with* $N_j \neq N_u$ **do**
    $Rest \leftarrow AvailableCPUOn(N_i) - CPU(VM_i) - epsilon$
    if $Rest_{min} < Rest$ **then**
      $Rest_{min} \leftarrow Rest$
      $DestinationNode(VM_i) \leftarrow N_j$
    end if
  end for
  if $DestinationNode(VM_i) \neq NULL$ **then**
    $AvailableCPUOn(DestinationNode(VM_i)) \leftarrow AvailableCPUOn(DestinationNode(VM_i)) - CPU(VM_i)$
    Migrate VM_i *from* N_u *to* $DestinationNode(VM_i)$
  end if
end for
if $NbOfVMsOn(N_u) = 0$ **then**
  Turn off N_i
end if

End

Algorithm 3 Size Up (Application Level)

Begin

for Each *tier* T_i *of the application* **do**
  if T_i *is saturated* **then**
    $NewVM \leftarrow$ *Allocation a new VM from the IaaS*
    Deploy and launch a new instance of replica of T_i *on* $NewVM$
    Reconfigure the loadbalancer in front of the T_i *tier*
  end if
end for

End

Algorithm 4 Size Down (Application Level)

Symbols:

- $NbOfReplicaOn(T_i)$: *The number of server replica on tier* T_i

Begin

for Each *tier* T_i *of the application* **do**
  if T_i *is underloaded and* $NbOfReplicaOn(T_i) > 1$ **then**
    $(ReplicaToRemove, VMToRemove) \leftarrow$ *A server replica and its VM*
    Reconfigure the loadbalancer in front of the T_i *tier*
    Stop $ReplicaToRemove$ *and terminate* $VMToRemove$ *from the IaaS*
  end if
end for

End

CPU load. The TUNe manager computes the average CPU load and when this metric reaches 100%, it requests a VM allocation from the IaaS, deploys and starts a new MySQL server instance on this VM. Notice that we choose this threshold in order to show the impact of overload of the MySQL tier on the application QoS (response time). Indeed, the application will still be overloaded during the addition time (startup of a new VM in the IaaS) of the MySQL server. To avoid this impact, one could either reduce this threshold or always keep a pool of unused VM in the running state. Likewise TUNe removes one MySQL server when its CPU load can be distributed on other replicas without provoking a MySQL tiers overload.

The overall management architecture for this scenario is as follows: two instances of TUNe are used for managing separately the two RUBiS applications (A and B) and implementing dynamic server provisioning. The IaaS does not implement any VM consolidation.

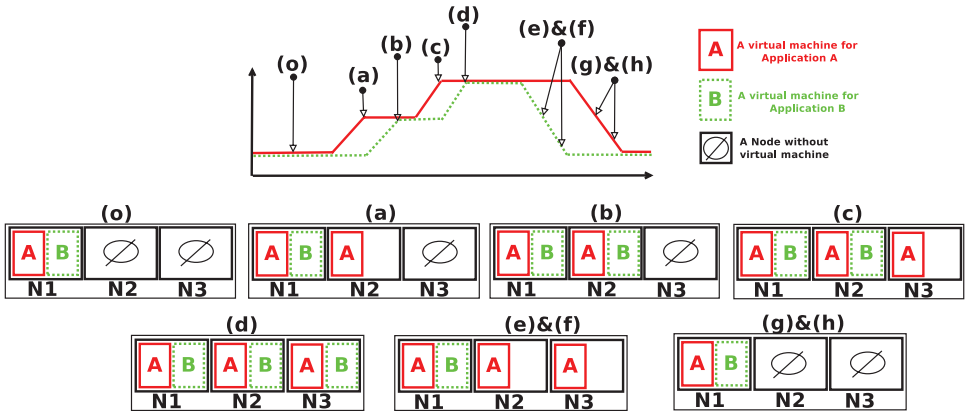

Fig. 7. Management at *application level* only: Load scenario and VM placement status.

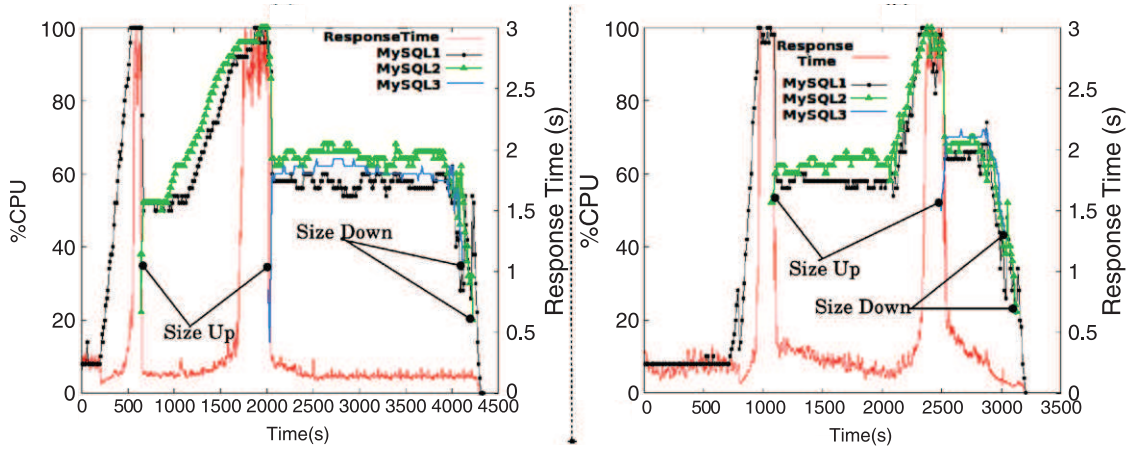

Fig. 8. Management at *application level* only: MySQL loads and response time of (a) application A and (b) application B.

We submit the same workload as in the previous experiment. Fig. 8 shows CPU loads on the nodes hosting MySQL servers and the RUBiS response time. These results show the reactivity of the autonomic system (TUNe) in order to ensure application QoS through dynamic resource allocation. These figures are interpreted as follows:

- In the first half of the workload (time (o)–time (d) in Fig. 7), the CPU load increases for both applications. As a result of TUNe's reconfigurations, MySQL servers are gradually added (times (a)–(d)) to keep the response time at its original level (less than 0.5 s). At the end of this phase (time (d)), each RUBiS application has 3 MySQL replicas. These *size up* operations are observed in Fig. 8(a) at times $T1 = 600$ s and $T2 = 2000$ s: the measured CPU load of the VMs reaches the maximum threshold and the response time also increases (about 3 s). The addition of a replica reduces the CPU load and also reduces the response time (at its minimal value). Notice that there is only one VM before T1, two VMs between T1 and T2, and three VMs after T2.
- After time (d) in Fig. 7, application B receives fewer requests, therefore, the number of replicas is reduced down to a single (on machine N1 at time (e) and (f)). This situation produces holes in the infrastructure (on nodes N2 and N3).
- A similar situation happens to application A: its load decreases. Its TUNe instance undeploys its MySQL instances and deallocates VMs on machines N2 and N3 (application A).
- At the end of this experiment (time (g) and (h) in Fig. 7), only machine N1 is running with two VMs, one for each application, similarly to the beginning of our scenario at time (o).

As can be seen in Fig. 8, the resource management policy at the application level effectively maintains QoS (i.e. keeps the response time low): MySQL instances are dynamically added and removed, according to the CPU load of this tier. This behavior ensures not only application response time but also a minimal number of VMs (and therefore cost reduction for the customer).

However, we can observe that at time (e) and (f), two MySQL instances (for application A) were running on two different VMs located on two different machines (N2 and N3). We lack here a consolidation mechanism to minimize the number of used machines, which is considered in the next section.

7. Resource management at both levels

This section describes a combination of the two strategies described in the previous sections to eliminate each solution's drawbacks.

7.1. Resource management policy

With a multi-level policy, we will both have dynamic sizing (the same algorithms as in the previous section) at the application level and a new form of consolidation at the IaaS level.

Since we have dynamic sizing, VM usage is optimized: VMs are effectively used or they will be removed by the TUNe instance at the application level. Therefore, it is not necessary at the virtualized level to reuse the consolidation policy based on CPU loads, and we can allocate Hard VMs. But as observed in Fig. 7 at time (e) and (f),

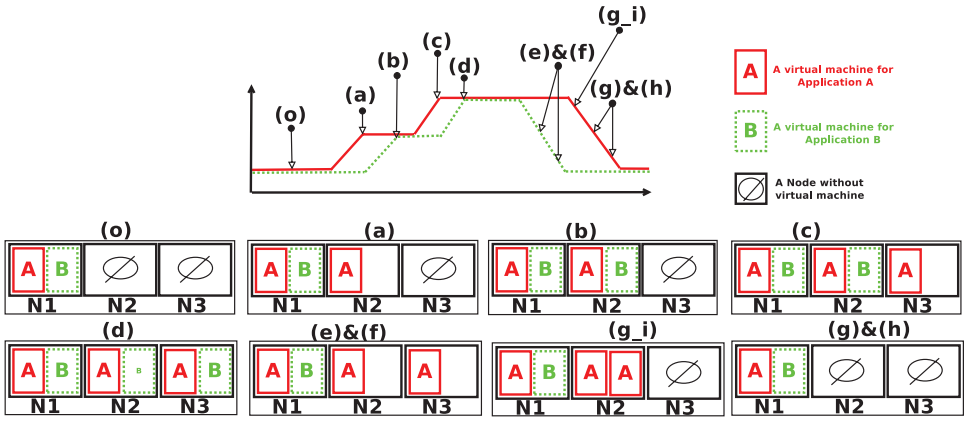

Fig. 9. Management at *both levels*: Load scenario and VM placement status.

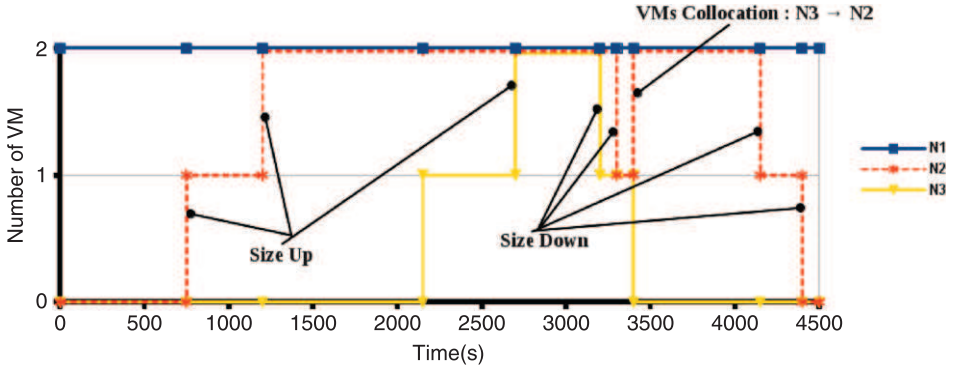

Fig. 10. Management at *both level*: VM allocation on nodes.

the application level policy can produce holes in the infrastructure. Thus, we implement a consolidation policy which migrates VMs to remove these holes. *However, this consolidation policy is not based on the CPU usage of VMs, but on a resource quota allocated to these VMs.*

Algorithm 5 describes this consolidation policy. In this algorithm and in our experiments, we allocate only one type of VM, i.e., all VMs have the same memory size and CPU quotas. It implies that we can host the same number of (Hard) VM on any machine in the IaaS. The *Scheduler* component of TUNe at the virtualized level tries to migrate the VMs from the machine with the minimum number of VMs. *MaxVM* is the number of VMs that a physical node may host and $NbOfVMsOn(N_i)$ the number of VM on each node. Such a consolidation can be performed as soon as $\sum NbOfVMsOn(N_i) \leq MaxVM * (MaxVM - 1)$. This condition is checked each time a VM is deallocated.

7.2. Evaluation

We use in this scenario three TUNe instances for managing the IaaS level and the two RUBiS applications. To distinguish between these TUNe instances, let *TUNeApp* be the TUNe instances running at the application level, and *TUNeVM* be the one assigned to manage the IaaS. These TUNe instances work independently, each handling the management operations at its level. TUNeApp's behavior (in terms of resource management) is the same as presented in Section 6: based on the average CPU load of the MySQL tiers, TUNeApp asks for the addition/removal of a MySQL server (running on a VM). TUNeVM's goal was described in the previous section: server consolidation based on VM allocated resources. This two-level management policy provides benefits for resource saving and power reduction, as shown in the experiment below.

The workload submitted to the two applications is the same as in the previous sections. Both RUBIS applications are started with one MySQL server. Each TUNeApp instance can allocate MySQL servers when necessary. Each MySQL VM requires half (in terms of CPU and memory quota) of an IaaS node during the experiment.

Fig. 9 shows the generated workload and predicted VM placements on physical nodes, according to our consolidation algorithm. Response times and VM loads in this scenario are similar to those of Section 6 (Fig. 8). The VM allocation status (per node) over time is presented in Fig. 10.

- Until time (d) ($T1 = 2800$ s), our experiment is in the allocation phase since the load increases for both applications. MySQL servers are gradually added.
- At time (e) and (f) ($T2 = 3300$ s), we observe the removal of two servers for application B on nodes N2 and N3, caused by the load reduction for application B.
- At tile (g_i) ($T3 = 3400$ s), the TUNeVM *Scheduler* analyzes the situation, migrates a VMA from N3 to N2 and frees N3.
- At time (d) ($T4 = 4200$–4400 s), the load reduction for application A provokes the removal of two MySQL servers.

This experiment shows that both-levels management cumulates the advantages of the two previous management strategies: it reduces the number of VMs in the IaaS and it consolidates these VMs on fewer machines (removing holes).

However, we can still have the problem identified in Section 5: the possibility to have on the same node several VMs hosting replicas from the same application tier. For example, at time (g_i),

Algorithm 5 VMs Consolidation (Both Level)

Symbols:

- *MaxVM* The number of VMs that a IaaS node may host
- *NbNode* The number of IaaS node which are running
- *NbOfVMsOn*(N_i): A function which returns the number of VMs running on node N_i

Begin

1: $N_{min} \leftarrow N_i$ *so that* *NbOfVMsOn*(N_i) *is the smallest*.
   # We try to free the node N_{min}.
2: **for Each** VM_i *on* N_{min} **do**
3: **for Each** *IaaS's node* N_i *with* $N_i \neq N_{min}$ **do**
4: **if** *NbOfVMsOn*(N_i) $<$ *MaxVM* **then**
5: *Migrate* VM_i *from* N_{min} *to* N_i
6: Break # Goto 2 (continue with another VM on N_{min}).
7: **end if**
8: **end for**
9: **end for**

End

two VMs hosting a MySQL replica for application A are located on N2. This problem comes from the assumption that the allocated VMs have a fixed size (the same size in our experiment) and that (i) the application level is not aware of VMs physical locations and (ii) the IaaS level is not aware of the application level replicas. A collaboration between both levels could allow the management of variable size VMs, in order to merge these two VMs (Fig. 10 (g_i)) into a single larger one (this is a perspective to this work).

8. Synthesis of experiments

This section highlights the main points of our described policies for resource management in virtualized infrastructure.

The management policy in our first scenario is implemented at the IaaS level only. Each application is deployed with a statically defined number of replicas (on Soft VMs) which is never changed (it is supposed to prevent peak loads). At runtime, VM migration is used to relocate or collocate VMs according to nodes' loads. This policy allows to free unused machines, but without dynamic sizing at the application level, the statically defined number of VMs implies that needless VMs are running in the IaaS upon underload conditions. Moreover, these VMs require memory, which limits the benefits from consolidation.

Our second scenario experiments the dynamic sizing policy at the application level (without any consolidation at the virtualized level). In this scenario, each application is deployed with an initial (minimal) number of replicas. Each replica is deployed on a separate Hard VM with guaranteed resources. At runtime, replicas are dynamically added and removed according to the load. This policy ensures that VM resources are effectively used (else they are removed). However, with several applications, when replicas are removed, it may leave holes (underused nodes) in the infrastructure. This application level management policy is therefore (obviously) lacking server consolidation at the IaaS level.

The third scenario is the combination of the two above policies: resource management is handled independently at the two levels. In this scenario, the CPU usage for each application is optimized at the application level thanks to dynamic sizing. Additionally, a consolidation policy triggers VM migrations to remove holes created by dynamic sizing. Moreover this consolidation policy is much simpler than in the first experiment, as it consolidates allocated VM quotas instead of VMs with varying CPU loads.

Finally, all consolidation algorithms we have described above can be integrated in to existing cloud platforms. For example, it is possible in the case of OpenNebula [15] throughout its *Scheduler* component.

9. Related work

Following our classification, we present in this section an overview of research works that have been conducted around resource management in the cloud computing area.

9.1. IaaS level management

Regarding resource management at the IaaS level, consolidation systems such as GreenCloud [16], Entropy [17,18] aim at saving energy in a hosting center using powering on/off nodes and VM migration. In the same vein, the Aneka [19] platform implements dynamic resource allocation in the context of hybrid clouds: it allocates new resources from a public cloud when those in a private desktop grid are overbooked. A deep thinking about VM consolidation problematic and solutions was done in [20]. This later formalizes the problem by defining two classes of solutions: deterministic algorithms and non deterministic algorithms. Regarding their formalism, our proposed algorithms are deterministic. In addition, [20] formalizes the cost of VM live migration in the case of a consolidation algorithm. The main purpose of [20] is not to propose a consolidation algorithm (as we have done), but to formalize the problematic. [21] studies VMs interference (the impact of co-locating several VMs which uses the same type of resources, on the same node) when applying a VM consolidation algorithm. It proposes a predictive model to avoid this impact. We do not take this into consideration. [22] presents a work in the same vein as [21], focusing on *n*-tier applications. [23] identifies and evaluates fourteen provisioning and allocation policies in the IaaS. It presents an overview of existing resource management policies and also evaluates their efficiency.

All these works address IaaS level consolidation policies. These VM consolidation algorithms could bring more benefits to cloud provider (in term of resource utilization) when they are combined with application sizing in the context of master–slave applications.

9.2. Application level management

Many research works have investigated dynamic resource allocation for application running in a virtualized hosting center environment. The major part of these works targets *n*-tier applications. [24] presents a dynamic allocation system for *n*-tier

applications, based on an autonomic computing system and a load balancer. This latter has a pool of allocated nodes and distributes the processes over this pool. The main drawback of [24] is that it relies on a single load balancer for all application, which can be a bottleneck. Also, its management policy is implemented within the load balancer which is very intrusive and not generic (our autonomic management system is not tied to a load balancing pattern and is therefore more generic). [25] proposed many strategies in a similar context as [24]: jobs distribution to a pool of VMs in a cloud infrastructure. These strategies are based on dynamic VM allocation/deallocation. New VMs are deployed when others are overloaded and they can be released when idle (our application level policy for J2EE applications is very similar). IBM's Oceano platform [26], as well as SmartScale [27], describes an automatic framework to scale applications using algorithms which meet those we have presented in this paper. [28] studies resource management at application level and focuses on the question of when to increase or decrease application replicas. To answer this question, it relies on trace analysis to model the behavior of the application in order to anticipate sizing actions. This method is well known in the literature under the term "feedback control LRU algorithm". Even if we have presented a proactive algorithm (based on actual traces), it is possible to implement [28]'s algorithm in our platform.

In summary, these research works about resource management in the cloud area are done regardless of each cloud actor. Moreover they work at the application level and manage only one application, while we address two cloud levels and the hosting of multiple customers applications.

9.3. Multi-level management

To the best of our knowledge, very few works have addressed dynamic resource management at both levels (application and IaaS) as presented in this paper. [29,30] have proposed a two-level resource management policy, but their mechanism of resource provisioning at the IaaS level is only based on the allocation of additional resources to VMs (vertical scalability). [31] proposes a solution to the coordination problem between VMs and the hosted applications when the resource availability have changed (more CPU or allocated memory for example). The [31]'s approach uses an hybrid solution (a feedback learning solution combine with a proactive solution) to prevent the reconfiguration of VMs and the applications they run. The latter are configured in a coordinated order: VMs before applications or vice versa. [32] presents and analyzes different resource management level in the cloud. The two levels we explore in this paper are mentioned in few words without a detailed solution. [33] proposes a model to coordinate different resource management policies involving the cloud customers and the provider. It defines a set of API and constraints allowing the customer to specialize resource allocation and services placement policies within the cloud. The limits of this work are: (1) the collaboration is done in one direction (from the customer to the provider); (2) it focuses on the customer's application elasticity policy (nothing about VM consolidation for instance). [34] describes a resource management framework for virtualized cloud environments. It implements both vertical and horizontal VM scalability in order to face workload evolution. The described framework needs to monitor specific application metrics (response time, throughput, etc.). Therefore, it is restricted to cloud platforms in which the cloud provider has access to the hosting applications. It has no interest in the complementarity of running several resource management policies at different cloud levels as we have presented in this paper. One of the first research work which investigates multi-level resource management in a virtualized cloud is [35]. As we claim in this paper, it proposes an autonomic system which automates resource management at two levels: application and IaaS. It also identifies two other levels: a local-level (VM on the same node), and a global-level (all VM running within the IaaS). Unlike our work, the different identified levels are all managed by the single resource management policy. This implies that the cloud customer and the provider are the same entity.

10. Conclusions and perspectives

Cloud computing is a recent trend where companies are externalizing their computing infrastructures in hosting centers, in order to reduce the cost of their IT. One of the main services that must be ensured in such a hosting center is resource allocation. For economic reasons, resources should be allocated (and deallocated) dynamically to the hosted applications according to their runtime load.

In this paper, we have investigated resource management strategies in the cloud computing context, with master–slave applications. We showed that consolidation of virtual machines at the IaaS level only with static application configuration incurs the overhead of needless collocated virtual machines. We then showed that dynamic application sizing only requires an IaaS consolidation in order to avoid holes (unused resources) on the IaaS physical machines, thus leading to a two-level resource management strategy.

Relying on a common application scenario, we justified such a two-level resource management policy that we prototyped and experienced in a private cloud infrastructure.

A perspective to this work is to improve this two-level resource management to further reduce collocated application replicas when the IaaS performs VMs consolidation. As described in this article, our algorithm relies on fixed size VM allocation and several VMs for the same tier may be required (according to the load) and collocated on a node. They could advantageously be replaced by a single bigger VM on that node. It becomes possible if the cloud provider has the knowledge of the architecture of customers' applications. In other words, it implies a collaboration between the two cloud actors (provider and customers), which also implies the upgrade of APIs commonly used in IaaS platforms.

We also plan to investigate the impact of VMs interferences in our resource management algorithms. Interferences occur when collocating on the same node several VMs which highly use shared resources whose reservation is not possible. These resources are disk and network IO.

Acknowledgments

This work is supported by two projects: (1) the French Fonds National pour la Societe Numerique (FSN) and Poles Minalogic, Systematic and SCS, through the FSN Open Cloudware project; and (2) the ANR INFRA (ANR-11-INFR 012 11) under a grant for the project ctrl-Green.

References

[1] A. Gulati, G. Shanmuganathan, A. Holler, I. Ahmad, Cloud-scale resource management: challenges and techniques, in: Proceedings of the USENIX Conference on Hot Topics in Cloud Computing, Portland, Oregon, USA, 2011, pp. 3–7.

[2] T.C. Chieu, A. Mohindra, A.A. Karve, A. Segal, Dynamic scaling of web applications in a virtualized cloud computing environment, in: Proceedings of the IEEE International Conference on e-Business Engineering, Macau, China, 2009, pp. 281–286.

[3] Righscale web site, visited on 2012, October in http://www.rightscale.com.
[4] C. Amza, A. Chanda, A. Cox, S. Elnikety, R. Gil, K. Rajamani, W. Zwaenepoel, E. Cecchet, J. Marguerite, Specification and implementation of dynamic web site benchmarks, in: IEEE International Workshop on Workload Characterization, Austin, TX, USA, 2002, pp. 3–13.
[5] Laurent Broto, Daniel Hagimont, Patricia Stolf, Noel Depalma, Suzy Temate, Autonomic management policy specification in Tune, in: Proceedings of the 2008 ACM Symposium on Applied Computing, Fortaleza, Ceara, Brazil, 2008, pp. 1658–1663.
[6] R. Buyya, C.S. Yeo, S. Venugopal, Market-oriented cloud computing: vision, hype, and reality for delivering it services as computing utilities, in: IEEE International Conference on High Performance Computing and Communications, Shanghai, China, 2009, pp. 5–13.
[7] C. Norris, H.M. Cohen, B. Cohen, Leveraging ibm ex5 systems for breakthrough cost and density improvements, in: Virtualized x86 Environments. White Paper, January 2011. ftp://public.dhe.ibm.com/common/ssi/ecm/en/xsw03099usen/XSW03099USEN.PDF.
[8] Alain Tchana, Suzy Temate, Laurent Broto, Daniel Hagimont, Autonomic resource allocation in a J2EE cluster, in: IEEE International Conference on Utility and Cloud Computing, Chennai, India, 2010.
[9] Mohammed Toure, Girma Berhe, Patricia Stolf, Laurent Broto, Noel Depalma, Daniel Hagimont, Autonomic management for grid applications, in: Proceedings of the Euromicro Conference on Parallel, Distributed and Network-Based Processing, Toulouse, France, 2008, pp. 79–86.
[10] Aeiman Gadafi, Alain Tchana, Daniel Hagimont, Laurent Broto, Remi Sharrock, N. De Palma, Energy-QoS tradeoffs in J2EE hosting centers, Int. J. Auton. Comput. September, (accepted in 2011) (in press).
[11] Qingyang Wang, Simon Malkowski, Deepal Jayasinghe, Pengcheng Xiong, Calton Pu, Yasuhiko Kanemasa, Motoyuki Kawaba, Lilian Harada, The impact of soft resource allocation on *n*-tier application scalability, in: Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium, Washington, DC, USA, 2011, pp. 1034–1045.
[12] Daniel Hagimont, Sara Bouchenak, Noel De Palma, Christophe Taton, Autonomic management of clustered applications, in: Proceedings of the IEEE International Conference on Cluster Computing, Barcelona, September, 2006, pp. 1–11.
[13] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, I. Touche, Grid'5000: a large scale and highly reconfigurable experimental grid testbed, Int. J. High Perform. Comput. Appl. 20 (2006) 481–494.
[14] P. Barham, B. Dragovic, K. Fraser, S. Hand, T.L. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the art of virtualization, in: Proceedings of the ACM Symposium on Operating Systems Principles, Bolton Landing, NY, USA, 2003, pp. 164–177.
[15] OpenNebula.org: the open source toolkit for cloud computing, visited on 2012, October. http://opennebula.org.
[16] L. Liu, H. Wang, X. Liu, X. Jin, W.B. He, Q.B. Wang, Y. Chen, Greencloud: a new architecture for green data center, in: Proceedings of the International Conference Industry Session on Autonomic Computing and Communications Industry Session, Barcelona, Spain, 2009, pp. 29–38.
[17] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, J. Lawall, Entropy: a consolidation manager for clusters, in: Proceedings of the ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, Washington, DC, USA, 2009, pp. 41–50.
[18] Pablo Graubner, Matthias Schmidt, Bernd Freisleben, Energy-efficient management of virtual machines in eucalyptus, in: Proceedings of the IEEE International Conference on Cloud Computing, Washington, DC USA, 2011, pp. 243–250.
[19] C. Vecchiola, R.N. Calheiros, D. Karunamoorthy, R. Buyya, Deadline-driven provisioning of resources for scientific applications in hybrid clouds with Aneka, Future Gener. Comput. Syst. 28 (2012) 58–65.
[20] A. Beloglazov, R. Buyya, Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers, Concurr. Comput.: Pract. Exper. (2012) 1397–1420.
[21] Qian Zhu, Teresa Tung, A performance interference model for managing consolidated workloads in QoS-aware clouds, in: Proceedings of IEEE International Conference on Cloud Computing, Honolulu, HI, USA, 2012, pp. 170–179.
[22] Simon Malkowski, Yasuhiko Kanemasa, Hanwei Chen, Masao Yamamoto, Qingyang Wang, Deepal Jayasinghe, Calton Pu, Motoyuki Kawaba, Challenges and opportunities in consolidation at high resource utilization: non-monotonic response time variations in *n*-tier applications, in: Proceedings of IEEE International Conference on Cloud Computing, Honolulu, HI, USA, 2012, pp. 162–169.
[23] David Villegas, Athanasios Antoniou, Seyed Masoud Sadjadi, Alexandru Iosup, An analysis of provisioning and allocation policies for infrastructure-as-a-service clouds, in: Proceedings of the IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Ottawa, Canada, 2012, pp. 612–619.
[24] H. Abdelsalam, K. Maly, R. Mukkamala, M. Zubair, D. Kaminsky, Towards energy efficient change management in a cloud computing environment, in: Proceedings of the International Conference on Autonomous Infrastructure, Management and Security: Scalability of Networks and Services, Enschede, The Netherlands, 2009, pp. 161–166.
[25] S. Genaud, J. Gossa, Cost-wait trade-offs in client-side resource provisioning with elastic clouds, in: Proceedings of the IEEE International Conference on Cloud Computing, Washington DC, USA, 2011, pp. 1–8.
[26] L. Zhang, D. Ardagna, SLA based profit optimization in autonomic computing systems, in: Proceedings of the international conference on Service oriented computing, New York, NY, USA, 2004, pp. 173–182.
[27] Sourav Dutta, Sankalp Gera, Akshat Verma, Balaji Viswanathan, SmartScale: automatic application scaling in enterprise clouds, in: Proceedings of IEEE International Conference on Cloud Computing, Honolulu, HI, USA, 2012, pp. 423–430.
[28] R. Hu, Y. Li, Y. Zhang, Adaptive resource management in PaaS platform using feedback control LRU algorithm, in: International Conference on Cloud and Service Computing, Las Vegas, Nevada, USA, 2011, pp. 11–18.
[29] J. Xu, M. Zhao, J. Fortes, R. Carpenter, M. Yousif, On the use of fuzzy modeling in virtualized data center management, in: Proceedings of the International Conference on Autonomic Computing, Jacksonville, Florida, USA, 2007, p. 25.
[30] Y. Song, H. Wang, Y. Li, B. Feng, Y. Sun, Multi-tiered on-demand resource scheduling for vm-based data center, in: Proceedings of the IEEE/ACM International Symposium on Cluster Computing and the Grid, Shanghai, China, 2009, pp. 148–155.
[31] Xiangping Bu, Jia Rao, Cheng-Zhong Xu, A model-free learning approach for coordinated configuration of virtual machines and appliances, in: Proceedings of the IEEE Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems, Singapore, Singapore, 2011, pp. 12–21.
[32] L.M. Vaquero, L. Rodero-Merino, R. Buyya, Dynamically scaling applications in the cloud, SIGCOMM Comput. Commun. Rev. 41 (2011) 45–52.
[33] Kleopatra Konstanteli, Tommaso Cucinotta, Konstantinos Psychas, Theodora A. Varvarigou, Admission control for elastic cloud services, in: Proceedings of the IEEE International Conference on Cloud Computing, Honolulu, HI, USA, 2012, pp. 41–48.
[34] Nicolas Bonvin, Thanasis G. Papaioannou, Karl Aberer, Autonomic SLA-driven provisioning for cloud applications, in: Proceedings of the IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Newport Beach, CA, USA, 2012, pp. 434–443.
[35] Ying Song, Hui Wang, Yaqiong Li, Binquan Feng, Yuzhong Sun, Multi-tiered on-demand resource scheduling for VM-based data center, in: Proceedings of the IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Shanghai, China, 2009, pp. 148–155.

Alain Tchana received his Ph.D. in Computer Science in 2011, at the IRIT laboratory, Institute National Polytechnique de Toulouse, France. Since November 2011 he has been a Postdoctor at the University of Grenoble (UJF/LIG). He is a member of the SARDES research group at LIG laboratory (UJF/CNRS/Grenoble INP/INRIA). His main research interests are in autonomic computing, Cloud Computing, and Green Computing.

Giang Son Tran is a Ph.D. student at the IRIT Laboratory, Institute National Polytechnique de Toulouse, France, funded by a scholarship from the Vietnamese government. He received his B.Sc. degree from the School of Information and Communication Technology, Hanoi University of Technology, Vietnam in 2003. His research interests are grid computing, autonomic management systems, mobile platforms and software engineering.

Laurent Broto is an Associate Professor at Polytechnic National Institute of Toulouse, France and a member of the IRIT laboratory, where he is member of a group working on operating systems, distributed systems and middleware. He received a Ph.D. from the Toulouse University, France in 2008. After a postdoctorate at the Oak Ridge National Lab, Oak Ridge TN, 2009, he took his position as an Associate Professor at Toulouse in 2009.

Noel DePalma received his Ph.D. in computer science in 2001. Since 2002 he was Associate Professor in computer science at University of Grenoble (ENSIMAG/INP). Since 2010 he has been a professor at Joseph Fourier University. He is a member of the SARDES research group at LIG laboratory (UJF/CNRS/Grenoble INP/INRIA), where he leads researches on Autonomic Computing, Cloud Computing and Green Computing.

Daniel Hagimont is a Professor at the Polytechnic National Institute of Toulouse, France and a member of the IRIT laboratory, where he leads a group working on operating systems, distributed systems and middleware. He received a Ph.D. from the Polytechnic National Institute of Grenoble, France in 1993. After a postdoctorate at the University of British Columbia, Vancouver, Canada in 1994, he joined INRIA Grenoble in 1995. He took his position as a Professor at Toulouse in 2005.