StopGap: elastic VMs to enhance server consolidation

Vlad Nitu*,†, Boris Teabe, Leon Fopa, Alain Tchana and Daniel Hagimont

IRIT-ENSEEIHT, Toulouse, France

SUMMARY

Virtualized cloud infrastructures (also known as IaaS platforms) generally rely on a server consolidation system to pack virtual machines (VMs) on as few servers as possible. However, an important limitation of consolidation is not addressed by such systems. Because the managed VMs may be of various sizes (small, medium, large, etc.), VM packing may be obstructed when VMs do not fit available spaces. This phenomenon leaves servers with a set of unused resources ('holes'). It is similar to memory fragmentation, a well-known problem in operating system domain. In this paper, we propose a solution which consists in resizing VMs so that they can fit with holes. This operation leads to the management of what we call elastic VMs and requires cooperation between the application level and the IaaS level, because it impacts management at both levels. To this end, we propose a new resource negotiation and allocation model in the IaaS, called *HRNM*. We demonstrate HRNM's applicability through the implementation of a prototype compatible with two main IaaS managers (OpenStack and OpenNebula). By performing thorough experiments with SPECvirt_sc2010 (a reference benchmark for server consolidation), we show that the impact of HRNM on customer's application is negligible. Finally, using Google data center traces, we show an improvement of about 62.5% for the traditional consolidation engines.

KEY WORDS: cloud computing; cooperation; resource management

1. INTRODUCTION

Nowadays, many organizations tend to outsource the management of their physical infrastructure to hosting centers, implementing the cloud computing approach. The latter provides two major advan-tages for end-users and cloud operators: flexibility and cost efficiency [1]. On the one hand, cloud users can quickly increase their hosting capacity without the overhead of setting up a new infras-tructure every time. On the other hand, cloud operators can make a profit by building largescale datacenters and by sharing their resources between multiple users. Most of the cloud platforms fol-low the Infrastructure as a Service (IaaS) model where users subscribe for virtual machines (VMs). In this model, two ways are generally proposed to end-users for acquiring resources: reserved and on-demand [2]. Reserved resources are allocated for long periods of time (typically 1–3 years) and offer consistent service, but come at a significant upfront cost. On-demand resources are pro-gressively obtained as they become necessary; the user pays only for resources used at each time. However, acquiring new VM instances induces instantiation overheads. Despite this overhead, on-demand resource provisioning is a commonly adopted approach because it allows the user to accurately control its cloud billing.

In such a context, both customers and cloud operators aim at saving money and energy. They generally implement resource managers to dynamically adjust the active resources. At the end-user level, such a resource manager (hereinafter *AppManager*) allocates and deallocates VMs according

*Correspondence to: Vlad Nitu, IRIT-ENSEEIHT, Toulouse, France.
†E-mail: vlad.nitu@enseeiht.fr

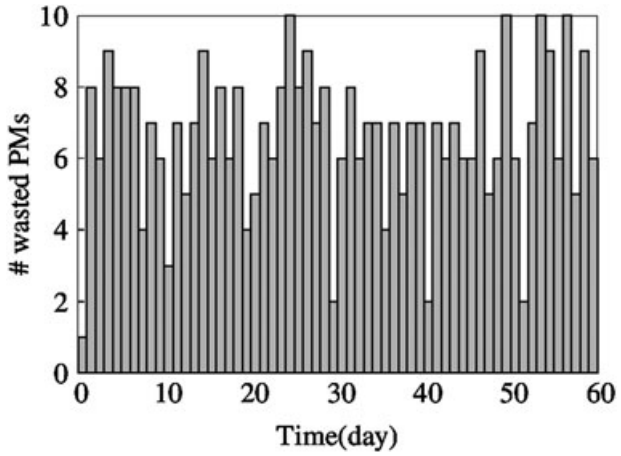

Figure 1. Resource wasting due to holes in a public Eolas cluster (a cloud operator) composed of 35 physical machines (PMs). Holes are aggregated and represented as entire wasted PMs. We can observe that an average of 6 PMs are misspent every day.

to load fluctuations [3]. Tools like Cloudify [4], Roboconf [5], Amazon Auto-scaling [6], and WASABi [7] can play that role. At the cloud platform level, the resource manager (hereinafter *IaaS-Manager*) relies on VM migration [8] to pack VMs atop as few physical machines (PMs) as possible. Subsequently, it leaves behind a number of 'empty' PMs which may be turned-off. This process is known as server consolidation [9]. Tools like OpenStack Neat [10], DRR/DPM from VMware [11], and OpenNebula [12] can play that role.

Although VM consolidation may increase server utilization by about 5–10%, it is difficult to actually observe server loads greater than 50% for even the most adapted workloads [2, 13, 14]. Because of various customer needs, VMs have different sizes (e.g., Amazon EC2 [15] offers more than 30 VM sizes) which are often incongruous with the hosting PM's size. This incongruity obstructs consolidation when VMs do not fit available spaces on PMs. The data center will find itself having a set of PMs which operate with long-term unused resources (hereinafter 'holes'). The multiplication of such situations raises the issue of PM fragmentation (illustrated in Section 2), which is a source of significant resource waste in the IaaS. Figure 1 presents the waste observed in a public Eolas [16] cluster[‡] composed of 35 PMs. The datacenter holes are aggregated and represented as entire wasted PMs. We can observe that an average of six PMs are misspent every day.

Virtual machines which consume a low amount of resource (hereinafter 'small' VMs) lead sometimes to a more efficient consolidation compared with VMs which consume a high amount of resource ('big' VMs). In order to take advantage of this fact, we introduce *StopGap*, *an extension which comes in support to any VM consolidation system*. It dynamically replaces (when needed) 'big' VMs with multiple 'small' VMs (seen as *VM split*) so that holes are avoided. StopGap imposes a novel VM management system that deals with *elastic VMs* (i.e., VMs whose sizes can vary during execution). However, current IaaS managers handle only VMs whose sizes are fixed during execution, thus we need to extend the traditional IaaS management model. To this end, we introduce a novel management model called Hybrid Resource Negotiation Model (HRNM), detailed in Section 4.

The main contributions of this article are the following:

1. We propose HRNM, a new resource allocation model for the cloud.
2. We propose StopGap, an extension which improves any VM consolidation system.
3. We present a prototype of our model built atop two reference IaaSManager systems (OpenStack [17] and OpenNebula [12]). We demonstrate its applicability with SPECvirt_sc2010 [18], a suite of reference benchmarks.

[‡]Eolas is our cloud computing partner.

4. We show that StopGap improves the OpenStack consolidation engine by about 62.5%.
5. We show that our solution's overhead is, at worst, equivalent to the overhead of First Fit Decreasing algorithms [19] underlying the majority of consolidation systems.

The rest of the paper is organized as follows. In Section 2, we introduce some notations, we motivate our new resource management policy, and we present its central idea. Section 3 defines the application type on which we tested our model. Section 4 presents in detail HRNM and its application to our reference benchmark. Section 5 presents StopGap while Section 6 evaluates both its impact and benefits. The paper ends with the presentation of related works in Section 7 and our conclusions in Section 9.

2. MOTIVATIONS AND MAIN IDEA

2.1. *Notations and definitions*

A data center is potentially wasting resources at a given time t if the following assertion is verified:

Assertion 1

$$\exists k \; s.t. \forall x \in \{CPU, memory, bandwidth\},$$
$$\sum_{j=1}^{m_{P_k}} booked_x(VM_j, P_k) \leqslant \sum_{i=1, i \neq k}^{n} free_x(P_i) \tag{1}$$

where

n	: total number of physical machines in the data center
VM_j	: VM number j
P_k	: PM number k
m_{P_k}	: total number of VMs on P_k
$booked_x(VM_j, P_k)$	: the amount of resource of type x booked by VM_j on P_k
$free_x(P_z)$	: the amount of resource of type x which is unbooked on P_z

In other words, a data center is wasting resources when there is at least one PM whose sum of booked resources by its VMs can be provided by the sum of the other PMs' holes. Figure 2 presents two states (top and bottom) of a data center with three PMs. According to our definition, in the first

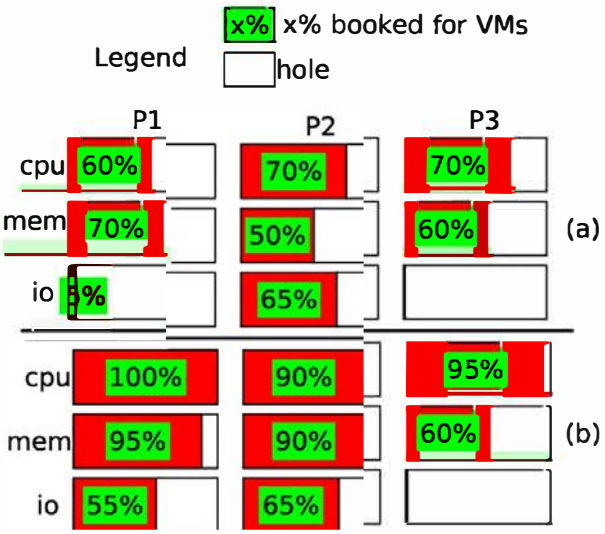

Figure 2. Illustration of resource waste in a data center: two states of the same data center are presented (a and b). VM, virtual machine. [Colour figure can be viewed at wileyonlinelibrary.com]

state, the data center wastes resources (k may be 1, 2 or 3). In the second case, we consider that the data center does not waste resources because Assertion 1 is not verified for CPU and memory.

2.2. Motivations

Resource waste is a crucial issue because of the tremendous energy consumed by today's data centers. Addressing this issue is beneficial on the one hand to cloud operators (about 23% of the total amortized costs of the cloud [20]). On the other hand, it is environmentally beneficial for the planet, as argued by Microsoft [21] (which proposes the 10 best practices to move in the right direction). Several research have investigated this issue, and the large majority of them [22, 23] rely on VM consolidation. The latter consists in dynamically rearranging (via live migration) VMs atop the minimum number of PMs. Thus, empty PMs are suspended (e.g., in sleep mode) or switched to a low power mode for energy saving.

Even if VM consolidation has proven its efficiency, it is not perfect for two main reasons: (i) VM consolidation is an nondeterministic polynomial time (NP)-hard problem; and (ii) in some situations, VM relocation is not possible even if Assertion 1 is verified. For illustration, let us consider our data center use case introduced in the previous section. We focus on the first state (Figure 2(a)) where resources are potentially wasted. As mentioned in the previous section, if we aggregate the P_2 and P_3 holes, we are able to provide the resources needed by all VMs which run on P_1. Therefore, one can think that by applying VM consolidation to this use case, P_1 could be freed.

> **Assertion 2**: The efficiency of any VM consolidation algorithm depends on two key parameters: VM sizes and hole sizes.

Returning to our first data center state (Figure 2(a)), we may consider two VM configurations which consume the same amount of resource on P_1:

- In Figure 3(a), we consider two identical 'small' VMs (VM_1 and VM_2). Each of them consumes 30% CPU, 35% memory, and 2.5% bandwidth from P_1. In this case, VM consolidation is able to migrate VM_1 to P_2 and VM_2 to P_3. At the end, P_1 may be turned-off (Figure 3(b)).
- In Figure 3(c) we consider that P_1 runs a single 'big' VM (VM'_1). VM consolidation is no longer efficient because neither P_2 nor P_3 is able to host VM'_1. It cannot fit in the available holes.

Such situations are promoted in a data center by the mismatch between VM sizes and holes. As presented in Section 1, Figure 1 shows that this issue is present in a real data center. In this paper we propose a solution which addresses this problem.

2.3. Basic idea

In the previous section, we exposed that the regular consolidation is difficult for 'big' VMs because they require big holes. A solution to this limitation could be to aggregate the holes using a distributed OS. However, the lessons learned from distributed kernels (such as Amoeba [24]) show that the reliability of these solutions is problematic. In this paper, we opt for an alternative approach which relies on two assumptions.

- (A_1) the *vertical scaling* capability of VMs: this is the virtualization system's capability to resize a VM (add/remove resources) at runtime. For instance, Xen and VMware (two widely used virtualization systems) provide this feature.
- (A_2) the distributed behavior of end-user applications: this is an application's capability to run atop a changeable number of VMs (*horizontal scaling*). Such applications are called elastic applications. They include the large majority of applications deployed within the cloud (e.g., internet services, MapReduce, etc.). For illustration, we focus in this paper on applications which follow the master–slave pattern.

Relying on these two assumptions, we propose a cooperative resource management system in which *the end-user allows the cloud manager to dynamically resize (vertical/horizontal scaling) his VMs so that a 'big' VM can be replaced by multiple small VMs without Service Level Agreement*

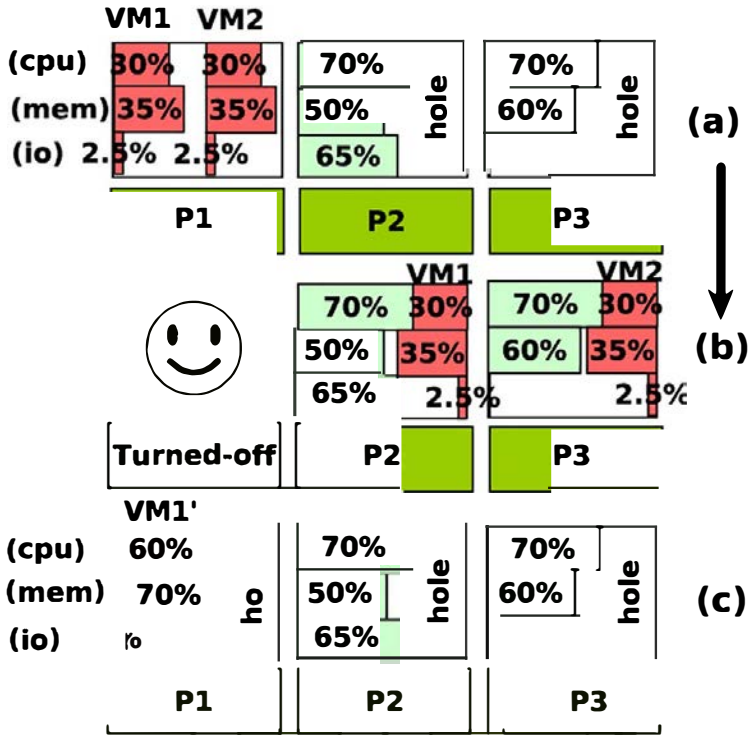

Figure 3. The efficiency of any virtual machine (VM) consolidation algorithm depends on two key parameters: VM sizes and hole sizes. From state (a), there is a possible VM consolidation which releases one PM; it is evidenced by the state transition (a)->(b). The consolidation is successful because of small VM sizes. In contrast, from state (c), VM consolidation is not possible because VMs are too big. [Colour figure can be viewed at wileyonlinelibrary.com]

(SLA) violation. For illustration, we apply our solution to the 'big' VM case in Figure 3(c). Firstly, we instantiate a new VM (VM_1) on P_1. Its size will be half of the 'big' VM size. Secondly, we scale down the 'big' VM (vertical scaling) to half of its size, resulting VM_2. Finally, we end up with the case of Figure 3(a) having two identical 'small' VMs.

Virtual machine resizing is not a common practice in nowadays cloud. Therefore, we propose a novel resource allocation and management model for the cloud. Before describing this model, we first present an overview of the master–slave pattern, the application type considered in our solution.

3. MULTI-TIER MASTER–SLAVE APPLICATIONS

As mentioned earlier, our solution is suitable for Multi-tier master–slave applications (hereinafter *MTMSA*). It is important to specify that MTMSA is one of the most prevalent architecture among Internet services. For instance, most applications from SPECvirt_sc2010 [18] and CloudSuite [25], two reference benchmarks for cloud platforms, follow this pattern. In this application type (Figure 4), a tier is composed of several replica (also called slaves) which all play the same role (e.g., web server, application server, and database). Each replica is executed by a single VM. In front of this set of slaves, lays a master VM, responsible for distributing requests to the slaves. The master is generally called loadbalancer because it implements a load balancing policy.

The main MTMSA advantage (which justifies its wide adoption) is the high flexibility of a tier (i.e., add/remove VMs according to the workload). After any change in a tier structure, the application has to be reconfigured. This process is usually automated by an autonomic-manager component (i.e., the $AppManager$) deployed with the application. The $AppManager$ is provided either by the Cloud (e.g., Amazon Auto-Scaling service), or by the customer (e.g., using an orchestration

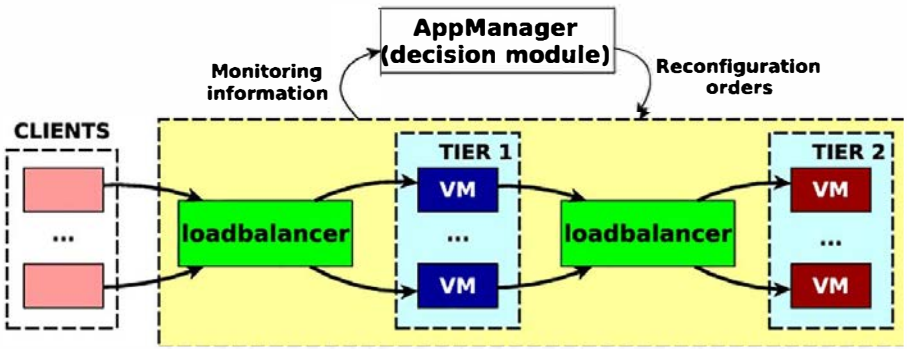

Figure 4. The architecture of an Multi-tier master-slave applications. VM, virtual machine. [Colour figure can be viewed at wileyonlinelibrary.com]

system like Cloudify [4] or Roboconf [5]). In this paper, we assume that the $AppManager$ is provided by the cloud. Generally, an $AppManager$ is responsible for:

- detecting a tier overload/underload and deciding how many VMs to add/remove (by sending instantiation/termination requests to the IaaSManager).
- invoking the loadbalancer reconfiguration in order to take into account a VM's arrival/departure.

4. A HYBRID RESOURCE NEGOTIATION MODEL

This paper improves VM consolidation thanks to the basic idea presented in Section 2.3. *Our solution is complementary to any VM consolidation system.* It requires a real collaboration between the AppManager (already provided by MTMSA) and the IaaSManager. This section presents the cooperative resource management model, which we propose. It can be considered from different perspectives: an extension of a PaaS or a hybrid IaaS-PaaS model. In this paper, we consider the latter case because it is the most general one.

4.1. Description of the model

In nowadays clouds, the resource negotiation model (between the customer and the provider) is based on fixed size VMs. We call it: the VM Granularity Resource Negotiation Model (hereinafter *VGRNM*). Figure 6 summarizes this model and illustrates its limitation in the perspective of VM consolidation. For instance, the sum of unused resources on PM_2 and PM_3 is greater than the needs of the large VM hosted on PM_4, but no consolidation system could avoid this waste.

Our model overcomes these limitations. To this end, it allows the IaaSManager to change both the number and the size of VMs, feature which is not provided by VGRNM. Thus, in addition to VGRNM, we need to define a new resource management model which allows resource negotiation at the granularity of an application tier. We call it: the Tier Granularity Resource Negotiation Model (hereinafter *TGRNM*). The HRNM (hybrid model) introduced earlier represents the aggregation between the traditional model (VGRNM) and our new model (TGRNM). Figure 5 graphically represents the negotiation phases of HRNM. They are summarized as follows:

- **(1)** Using VGRNM, the customer deploys and starts his AppManager, which exposes a web service. Through it, the AppManager is informed about any resource changes (e.g., after a VM resize). Finally, the customer registers the AppManager endpoint with the IaaSManager.
- **(2)** The customer enters in what we call the 'subscription phase'. An application subscription is formalized as follows: $A = \{t_i(\#cpu, \#mem, \#io) and strategy | 1 \leqslant i \leqslant n\}$, where A represents a request to the provider (see Figure 5 left), n is the total number of tiers, t_i represents the i^{th} tier, $(\#cpu, \#mem, \#io)$ is the tier size, and strategy represents the allocation model (TGRNM or VGRNM).

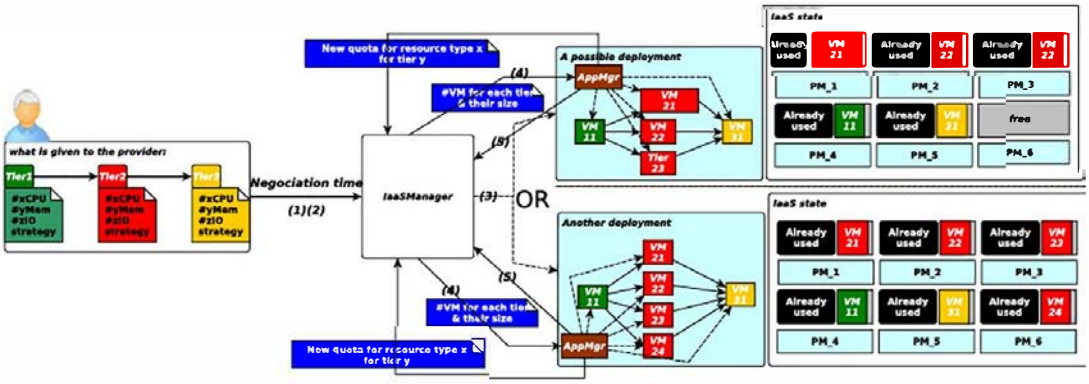

Figure 5. The general behavior of our solution. PM, physical machine; VM, virtual machine. [Colour figure can be viewed at wileyonlinelibrary.com]

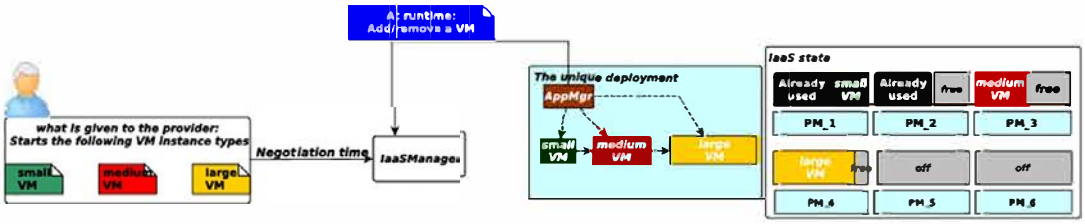

Figure 6. The traditional functioning of a cloud platform. The resource negotiation model is based on fixed size virtual machines (VMs) (small, medium, large, etc.) requested by the end-user. The cloud operator has no information about the application type (its architecture) deployed within VMs. Furthermore, any modification of the application is initiated by the AppManager (VM addition or removal) according to workload fluctuations. This inflexibility is at the heart of VM consolidation limitations: see resource waste on physical machine (PM)_2 and PM_3. [Colour figure can be viewed at wileyonlinelibrary.com]

- **(3)** From these information, the IaaSManager computes and starts the number of VMs needed to satisfy each tier. The first advantage (resource saving) of our solution can be observed during this step. Indeed, VM instantiation implies VM placement: which PMs will host instantiated VMs. An efficient VM placement algorithm avoids resource waste. In comparison with the traditional model in which VM sizes are constant and chosen by end-users, our model avoids PM fragmentation (the multiplication of holes). For instance, PM_2 and PM_3 in Figure 6 (the traditional model) have unused resources which would have been filled in our model (as shown in Figure 5 right).
- **(4)** When VMs are ready, the IaaSManager informs the AppManager about the number and the size of VMs for each tier so that the application can be configured accordingly (e.g., load balancing weights).
- **(5)** The AppManager informs back the IaaSManager when the application is ready. Resource changes are envisioned only after this notification.

As mentioned earlier, the traditional model (VGRNM) is still available in our solution because it could be suitable for some tiers. For instance, the MTMSA entry point (i.e., loadbalancer) needs a static well-known IP address, thus a single VM. More generally, our solution is highly flexible in the sense that it is even possible to organize a tier in two groups so that each group uses its own allocation model (Section 6 presents a use case). The next section presents an application of our model to a well-known set of cloud applications.

4.2. Application of the model

SPECvirt_sc2010 [18] is a reference benchmark which has been used for evaluating the performance of the most common cloud platforms. It is composed of three main workloads which are the patched versions of more specific benchmarks: SPECweb2005 [26] (web application),

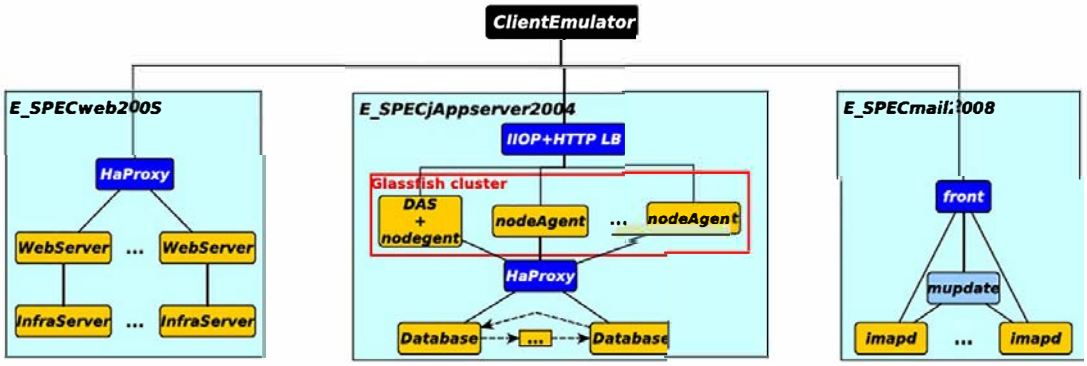

Figure 7. Architecture of E_SPECvirt. [Colour figure can be viewed at wileyonlinelibrary.com]

Table I. VGRNM and TGRNM: which model is appropriate to each E_SPECvirt tier?

	E_SPECweb	E_SPECjAppserver	E_SPECmail
VTGRNM	HaProxy, InfraServer	HaProxy, DAS	Front, Mupdate
TGRNM	Web	Glassfish, MySQL	Imapd

SPECjAppserver2004 [27] (JEE application), and SPECmail2008 [28] (mail application). SPECvirt_sc2010 also provides a test harness driver to run, monitor, and validate benchmark results. We relied on SPEC-virt_sc2010 in order to target the most popular cloud applications. For the needs of this paper, we enhanced SPEC-virt_sc2010 by implementing the elasticity of each tier. This new version is called E_SPECvirt, and it is composed of E_SPECweb2005, E_SPECjAppserver2004, and E_SPECmail-2008. Figure 7 presents the new architecture. E_SPECweb2005 comprises one or more Apache [29] web servers with loadbalancing assured by HaProxy[30]. E_SPECjAppserver2004 is composed of a Glassfish [31] cluster with loadbalancing archived by both HaProxy (for HTTP requests) and Domain Administration Server (DAS) registry (for Internet Inter-Orb Protocol (IIOP) requests). HaProxy also provides loadbalancing for MySQL databases. To ensure consistency, all MySQL servers leverage a master–master replication [32]. Update requests received by a MySQL server are replicated to the others in a cyclic way. E_SPECmail2008 is achieved by Cyrus IMAP [33]. The latter provides three software types: a loadbalancer (called front), a database server, which contains information about the location of all mailboxes (called mupdate), and multiple imapd slaves which serve IMAP requests.

Table I shows which HRNM submodel is suitable for each E_SPECvirt tier. VGRNM is used both for loadbalancers and for some software such as InfraServer, DAS, and Mupdate, which need to be known in advance throughout a unique static IP address (thus a single VM). All other tiers are provisioned using TGRNM.

5. IMPLEMENTATION OF THE MODEL

In the cloud, a customer can request resources both at application subscription time or at runtime. There are two types of cloud actions at runtime:

- (C_1) the adjustment of both the number and the size of VMs while keeping the corresponding tier to the same size.
- (C_2) the adjustment of a tier size in response to workload variation.

C_1 operation types are initiated by the IaaSManager while C_2 operation types are initiated by the AppManager. The 'subscription phase' can be seen as a C_2 operation: increase the tier size start-

ing from zero. A runtime cloud action is taken only if the application performance insured by the provider (i.e., the SLA) is respected. The procedure used to ensure the SLA is presented in the succeeding text.

5.1. Service Level Agreement enforcement during virtual machine split

One of the main goals of a cloud operator is to save resources. Thus, every time Assertion 1 is verified, it considers that there should be a better consolidation. In this respect, the IaaSManager tries to restructure application tiers by replacing 'big' VMs with 'smaller' VMs (VM split). The main objective of this operation is to improve VM consolidation (free as much PMs as possible). On the other hand, the customer is rather interested in the performance of its application. *There are cases where even if Assertion 1 is verified, the provider cannot split a VM*. These circumstances are promoted by two main factors. First, there is often a non-linear dependence between the performance and the amount of resource. For instance, a 2 GB VM may not perform two times better than a 1 GB VM. Second, there is always an overhead introduced by VM's operating system (OS) footprint. For an accurate VM split, we need to find a metric which exposes well the application performance. A suitable choice for our MTMSA seems to be the maximum application throughput (e.g., requests/sec for a web server). Based on this metric, we can safely split the VM without affecting the customer. For example, a customer will be satisfied with both, a single VM capable of 200 req/s or 2 VMs, each one capable of 100 req/s, considering that the streams are aggregated by the loadbalancer. To convert from resources to throughput, we introduce a function called $s2t_{tier}$ (size to throughput for a given tier). It takes as input a hole (#cpu,#mem,#io) and returns the throughput that a corresponding VM will deliver. The function is provided either by the customer[§] or by the provider. If the customer does not have such information, the provider (IaaSManager) computes the function by relying on Quasar [2]. The latter dynamically determines application throughput based on performance monitoring counters and collaborative filtering techniques. The estimation of $s2t_{tier}$ is beyond the scope of this paper.

```
1 // This function is invoked at the end of each VM consolidation round
2 void consolidationExtension (...) {
3  S1={PMs with holes}
4  thr_holes − "the sum of throughput of all holes"
5  choose P from S1 so that P has the biggest hole
6  thr_P − "the sum of throughput of all P's VMs"
7  if (thr_P > thr_holes)
8   return
9   foreach (VM v on P){
10    determine tierName of v
11    thr_v := s2t_tierName(sizeOf(v))
12    foreach (P_i ∈ S1 ∩ {P}) {
13     thr_hole := s2t_tierName(sizeOfhole(P_i))
14     if (thr_v<thr_hole){
15      resize v to thr_v
16              notify changes to AppManager
17              migrate v to P_i
18              break
19     } else {
20      enlarge or instantiate a VM in this hole
21              thr_v := thr_v − thr_hole
22     }
23     update S1
24    }
25  }
26  Switch−off PMs without VMs
27 }
```

Figure 8. The StopGap algorithm. PM, physical machine; VM, virtual machine. [Colour figure can be viewed at wileyonlinelibrary.com]

[§]Customers may have such information because they need to predict how their applications will perform in a given VM.

5.2. *Resource management of type C_1*

While HRNM can improve VMs' resource assignment at application subscription time, holes may also show up during runtime (e.g., a VM termination/migration). In order to address this issue, we introduce a VM consolidation extension called StopGap. Figure 8 presents in pseudo code the StopGap algorithm. It is complementary to the consolidation system which already runs within the IaaS. The only thing to do is to immediately invoke it after each VM consolidation round. For simplicity reasons, we are not presenting the pseudo code related to synchronization problems. In the real code version, we used locks to avoid holes contention (Section 6.1). Figure 9 illustrates the algorithm on a simple use case: the restructuring of the web server tier in Specvirt_sc2010. The StopGap algorithm is interpreted as follows. The reader can follow in parallel the illustration in

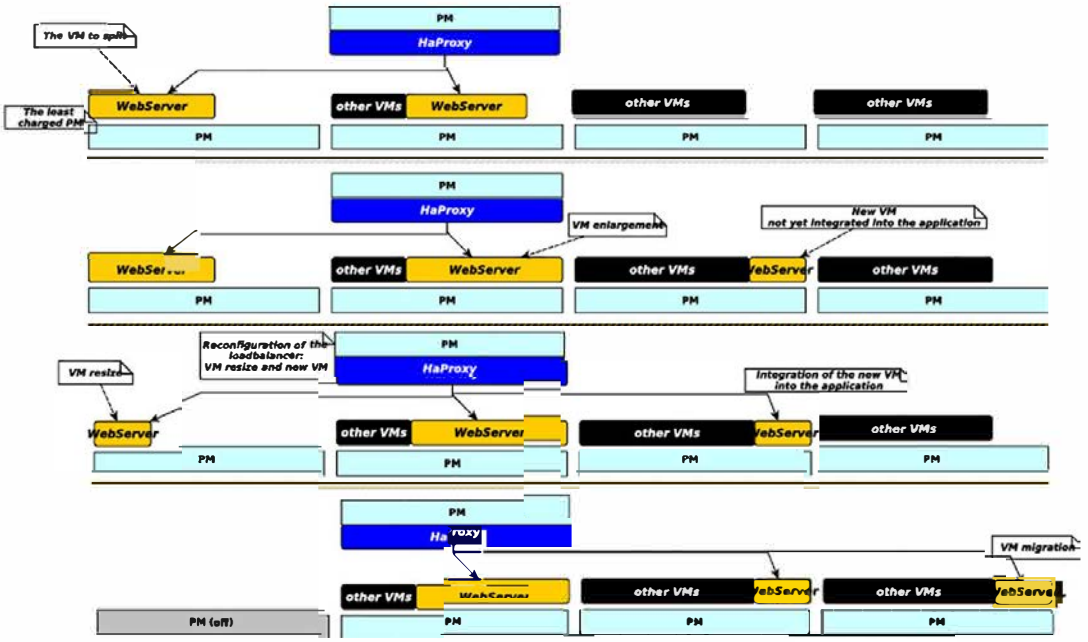

Figure 9. Illustration of StopGap on E_SPECweb2005. PM, physical machine; VM, virtual machine. [Colour figure can be viewed at wileyonlinelibrary.com]

```
1 // decrease tierName by Δ
2 S1={PMs which run a VM belonging to the tierName}
3 thrΔ = s2t_tierName(Δ)
4 decLabel:
5 Let v be "the smallest VM of tierName in S2"
6 thrv := s2t_tierName(sizeOf(v))
7 if (thrv==thrΔ){
8         record v for termination
9 }else if (thrv < thrΔ){
10        record v for termination
11        thrΔ := thrΔ − thrv
12        remove P from S2
13        goto decLabel
14 }else{
15        shrink v until s2t_tierName(sizeOf(v)) == thrv − tΔ
16 }
17 notify changes to the AppManager
18 when(ack is received){
19        terminate recorded VMs
20        free empty PMs
21 }
```

Figure 10. Tier size decrease algorithm. PM, physical machine; VM, virtual machine. [Colour figure can be viewed at wileyonlinelibrary.com]

```
1 // increase tierName by Δ
2 S1={PMs with holes}
3 thrΔ = s2t_tierName(Δ)
4 foreach (Pi ∈ S1){
5         thr_hole := s2t_tierName(sizeOfhole(Pi))
6         enlarge or instantiate a VM in this hole
7         thrΔ := thrΔ − thr_hole
8 }
9 incLabel:
10        turn−on a new PM P
11        instantiate a new VM v on P
12        thr_v := s2t_tierName(sizeOf(v))
13        thrΔ := thrΔ − thr_v
14        if (thrΔ > 0)
15                goto incLabel
16 notify changes to the AppManager
```

Figure 11. Tier size increase algorithm. PM, physical machine; VM, virtual machine. [Colour figure can be viewed at wileyonlinelibrary.com]

Figure 9: top-down. First, we choose the least charged PM (line 5) (noted P). If the data center holes are unable to provide the necessary throughput for P's VMs (line 7), no application restructuring can be done without performance loss. Otherwise, we take a VM v from P (line 9). We iterate over the remaining PMs from S_1 (line 12), and we start to reconstruct v in the holes(lines 20-21). If a VM of the same tier exists on P_i, we prefer to enlarge it instead of instantiating a new VM. Each time, we subtract the new VM's throughput from the throughput of v (line 21). When we find a hole which can provide the remaining throughput, we migrate v (lines 15–18). Notice that the reconfiguration of the application during VM reconstruction is only performed once all generated VMs are ready. By doing so, there is no wait time related to VM instantiation.

5.3. Resource management of type C_2

Because of workload variations, the AppManager may request a change in a tier's capacity/size. Figures 10 and 11 present in pseudo code the algorithms to shrink/enlarge a tier. It works as follows. The AppManager communicates to the IaaSManager the desired tier variation (Δ). Concerning the tier downscale (Figure 10), the IaaSManager prioritizes VM termination (line 8, 10) rather than shrinking a group of VMs (line 15). Thus, the overhead caused by VM's OS footprint is minimized. Regarding the tier enlargement (Figure 11), the priority is placed on resizing (vertical up-scaling) the existing VMs. If at the end, the request is not completely satisfied, a set of VMs are instantiated according to available holes (line 6). If all holes are filled up and the request is still not completely satisfied, PMs are switched-on, and new VMs are instantiated atop them (line 10–11). The IaaSManager always informs back the AppManager about the operations it has performed (i.e., new size for old VMs, new VMs and their size, or terminated VMs). Subsequently, the AppManager answers with an ACK message. The IaaSManager only terminates VMs upon receiving that message. This prevents the termination of a VM which is still servicing requests.

6. EVALUATIONS

In order to test our approach, we performed two evaluation types. The first type evaluates our solution impact on customer applications, provided by SPECvirt_sc2010 [18] (presented in Section 4.2). The second evaluation type focuses on VM consolidation improvements.

6.1. Experimental environment

The first type experiments were performed using a prototype implemented within our private IaaS. It is composed of 7 HP Compaq Elite 8300, connected with a 1 Gbps switch. Each node is equipped with an Intel Core i7-3770 3.4GHz and 8 GB RAM. One node is dedicated to management systems (IaaSManager, NFS server, and additional networking services: Domain Name System, Dynamic

Host Configuration Protocol). The others are used as resource pool. To show the generic facet of our solution, the prototypes have been implemented for two reference IaaSManger systems: OpenStack [17] and OpenNebula [12]. Both systems are virtualized with Xen 4.2.0. The integration of our solution with these systems is straightforward. We have implemented the resource negotiation model on top of both OpenStack and OpenNebula public APIs. Concerning VM consolidation, OpenStack relies on OpenStack Neat [10]. It is an external and extensible framework which is provided with a default consolidation system. Our solution requires a minor extension to OpenStack Neat. We only extended its 'global manager' component, which implements the consolidation algorithm. Two modifications were necessary: one Line of Code (LoC) at the end of the consolidation algorithm to invoke StopGap (Figure 8), and about 5 LoCs for locking PMs whose VMs are subject to resize. This prototype is used to evaluate the impact of our resource allocation model. Concerning OpenNebula, it does not implement any dynamic VM consolidation module. However, it is built so that the integration of a consolidation engine is elementary. In OpenNebula, the single component which we patched is the 'Scheduler'.

6.2. Impact on end-user's applications

In our solution, two new operation types can impact the performance of end-user applications:

- Vertical and horizontal scaling. By leveraging HRNM, the IaaSManager may combine vertical scaling (VM size adjustment) and horizontal scaling (add/remove a VM) to dynamically restructure an application tier. These operations are the basis for both VM split and VM enlargement.
- Application reconfiguration: VM spliting and enlargement require the adaptation/reconfiguration of the application level (e.g., weight adjustment).

6.2.1. Impact of vertical and horizontal scaling. The influence of each operation is evaluated separately. Figure 12 presents the experiment results. In Figure 12(a), we can note that the time taken to instantiate or terminate VMs is quasi constant regardless the number of VMs (about 20 s to instantiate and 2 s to terminate). This is due to the parallel VM instantiation/termination. Notice that neither VM instantiation nor VM termination impact applications which run on the same machine because these operations do not require high amount of resource for completion.

Concerning vertical scaling, we evaluated addition/removal of each resource type individually. We evaluated the time taken to make added resources (respectively removed resources) available (respectively unavailable) inside the VM. As reported in Figure 12(b), vCPU addition or removal time increases almost linearly with the number of vCPUs. This is explained by the fact that any adjustment in the number of vCPUs triggers the sequential execution of a set of watchers (according to the number of vCPUs). Notice that vCPU removal costs about 20 times more than addition.

Similar results have been reported for the main memory. Its shrinking corresponds to the time taken by the VM to free memory pages. This operation is triggered by a balloon driver which resides within the VM. Concerning memory addition, it corresponds to the time taken by the hypervisor to

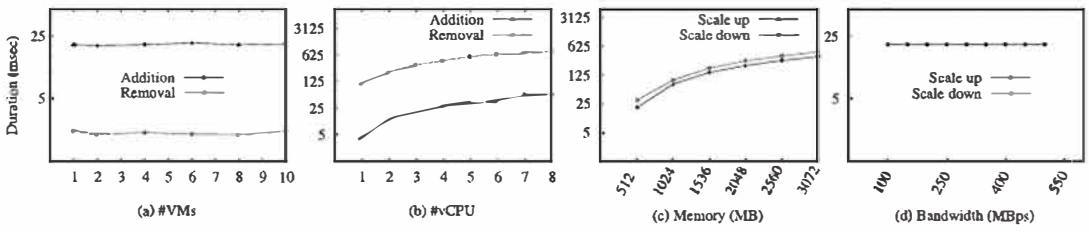

Figure 12. Horizontal and vertical scaling durations (N.B.: log-scale y-axes). Virtual machine (VM) instantiation or termination time is quasi constant regardless the number of VMs. vCPU addition or removal time increases almost linearly with the number of vCPUs. Similar results are observed for the main memory. Bandwidth adjustment always uses the same duration.

Table II. E_SPECvirt *AppManagers* reconfiguration algorithms.

```
// AppServer: VM addition
Update haproxy.cfg and reload it
Update glassfish .env and reload it
Start a new Glassfish node agent
Start a new Glassfish server
// AppServer: VM removal
Update haproxy.cfg and reload it
Update glassfish .env and reload it
Update domain.xml and reload DAS
Stop Glassfish server
Stop Glassfish node agent
// AppServer: VM resize
Update haproxy.cfg and reload it
Update glassfish .env and reload it
// web: VM addition/removal/ resize
Update haproxy.cfg and reload it
//mail: VM addition
Update the front server
Reorganize (via migration) mailboxes according to backend's size
```

```
// DB: VM addition
Start MySQL with specdb database
DB pre-sync
Lock all the active DBs
Execute the final rsync
Unlock the DBs
Update the circular relationship of MySQL slaves
Update haproxy.cfg and reload it
// DB: VM removal
Update haproxy.cfg and reload it
Update MySQL slaves relationship
// DB: VM resize
Update haproxy.cfg and reload it
// mail: VM removal
Migrate mailboxes from the removed server
Update the front server
// mail: VM resize
Reorganize mailboxes according to backend's size
```

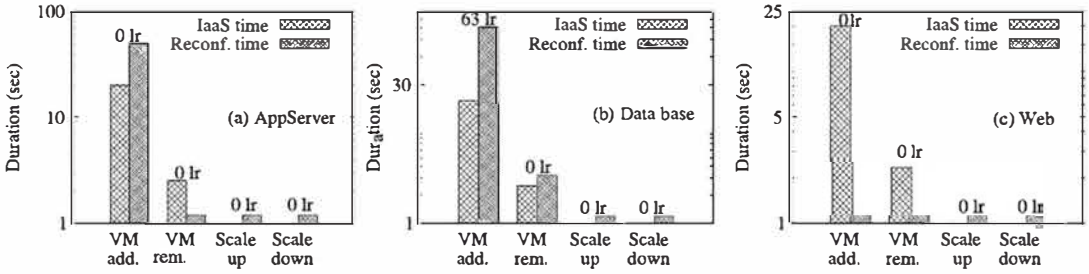

Figure 13. Horizontal and vertical scaling impact on (a)App Server, (b)DataBase (DB) Server and (c)Web Server (N.B.: log-scale y-axes). Except the addition of a new data base server, no tier suffers from our solution. VM, virtual machine.

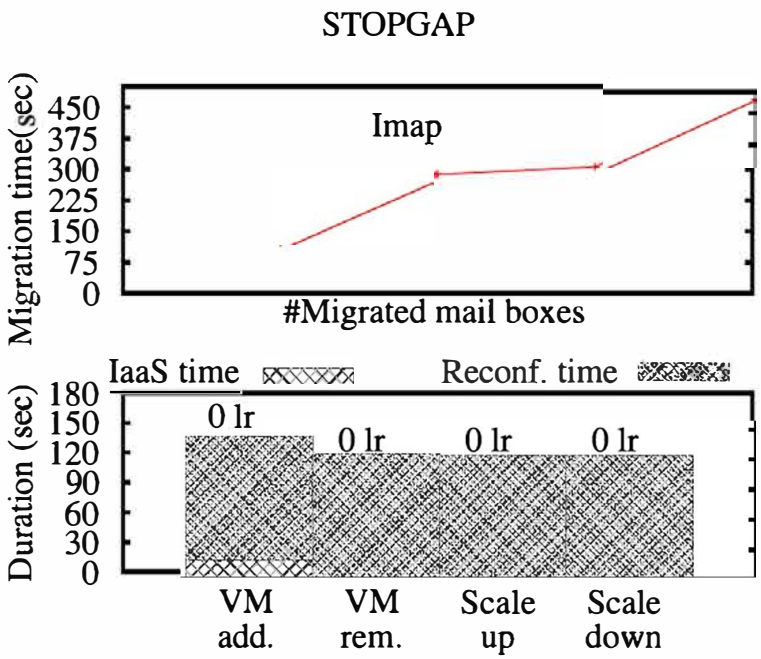

Figure 14. Impact of horizontal and vertical scaling on Imap Server. No request is lost during the reconfiguration. The migration time increases almost linearly with the number of migrated mailboxes. VM, virtual machine. [Colour figure can be viewed at wileyonlinelibrary.com]

both acquire machine memory pages (which is straightforward in the context of our solution because it uses holes) and update the memory page list used by the target VM. Last but not least, bandwidth adjustment always implies a constant time. Because Xen does not manage bandwidth allocation, we relied on tc [34], a Linux tool which quickly takes into account the bandwidth adjustment. Every time a packet is sent or received, tc checks if the bandwidth limit is reached. Thus, a bandwidth adjustment is immediately taken into account.

6.2.2. Impact of dynamic reconfigurations. The second set of experiments evaluate both the application reconfiguration time and the consequences of this operation. The adopted impact indicator is the number of lost requests during the reconfiguration (noted lr). For each experiment, the workload is chosen so that VMs are saturated. Table II presents in a pseudo-code the reconfiguration algorithms we have implemented for each tier. Figures 13 and 14 report the results of this second set of experiments, interpreted as follows. The number of lr is shown atop each pair of histogram bars.

- *Application server tier* (Figure 13(a)). Except the integration of a new VM, which takes some time (the first two bars), the reconfiguration of the application server tier is straightforward. During this operation, no request is lost. Our solution does not incur major issues for this tier.
- *Database tier* (Figure 13(b)). A new database integration within the application is relatively expensive (the first two bars). During this operation, the database tier is out of service for a few moments because of synchronization reasons. This is the only situation which leads to some lost requests. Therefore, our solution could become negative for E_SPECj-Appserver2004 if the addition of new MySQL VMs occurs frequently. This problem does not appear when removing or vertically scaling a database VM because no synchronization is needed. In these cases, only the loadbalancer needs to be reconfigured.
- *Web tier* (Figure 13(c)). The web tier reconfiguration is straightforward because it only requires a loadbalancer update. Our solution does not impact this tier.
- *Mail tier* (Figure 14). The time taken at the application level to reconfigure the mail tier (Cyrus IMAP) is almost the same regardless the reconfiguration option (Figure 14 bottom). In any case, the same number of mail boxes needs to be migrated. Because of the mailbox live migration implemented by Cyrus, no request is lost during the reconfiguration. The migration time increases almost linearly with the number of migrated mailboxes (Figure 14 top).

Impact of multiple reconfigurations. We tend to conclude from the aforementioned experiments that the impact of a single reconfiguration is almost negligible. However, the multiplication of these actions on a group of VMs belonging to the same application could be harmful. Figure 15 presents the normalized performance of each specific benchmark when the number of reconfiguration oper-

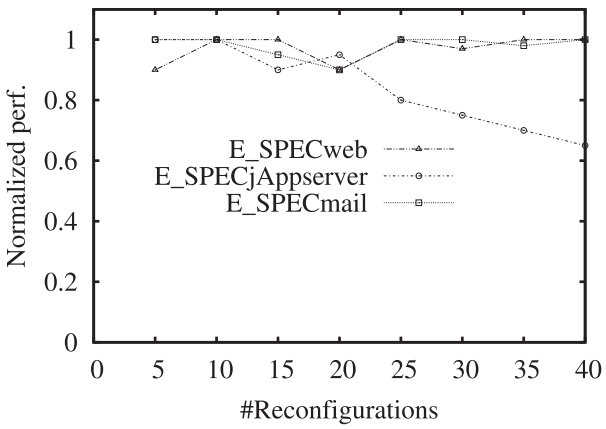

Figure 15. Impact of performing several reconfiguration operations. Both E_SPECmail and E_SPECweb are relatively little impacted by the multiplication of reconfiguration operations. In contrast, E_SPECjAppserver's performance starts to degenerate after about 20 reconfigurations. The degradation comes from the synchronization of the data base tier, which requires a little downtime of the application.

ations varies. We can note that both E_SPECmail and E_SPECweb are relatively little impacted by the multiplication of these operations. This is not the case for E_SPECjAppserver whose performance starts to degenerate after about 20 reconfigurations. The number of additions of new database VMs increases. To minimize this impact, the algorithm presented in Figure 8 has been improved for fairness. The reconfiguration operations which are performed in order to improve VM consolidation are fairly distributed among cloud applications. Thus, PMs whose VMs are subject to split are fairly chosen.

6.2.3. Synthesis. In comparison with horizontal scaling, vertical scaling globally provides better results regarding reconfiguration duration and performance degradation. Several reasons explained that. First of all, reconfiguration operations required to be performed at application level after a vertical scaling are most of the time less complex than those needed after an horizontal scaling (see algorithms in Table II). Secondly, resource (un)plug-in is faster (in mere microseconds) than VM instantiation/termination (in mere seconds). These two options are showcased in our solution.

6.3. Resource saving and scalability

Resource saving The main goal of our contribution is resource saving. For this evaluation type, we rely on Google data center traces obtained from [35]. Before presenting the results, let us firstly introduce how we interpreted Google traces. They represent the execution of thousands of jobs monitored during 29 days. Each job is composed of several tasks, and every task runs within a container. For each container, we know the amount of resource used by the job and the PM on which it is executing. We considered a job as a customer application where its number of tasks correspond to the number of tiers. Therefore, a container is seen as the VM allocated during the first resource allocation request. The total number of PMs involved in these traces is 12,583, organized into eleven types. For readability, we only present in this paper the analysis of a subset of these traces. It includes up to 7669 PMs and 82,531 VMs. Figure 16 summarizes its content. We evaluated how the StopGap extension may improve OpenStack Neat (OSN for short). The number of freed PMs is compared when OSN runs in three situations: alone (noted 'OSN'), in combination with our solution when every second tier leverages StopGap (noted 'OSN+(1/2)StopGap'), and in combination with our solution when all tiers leverage StopGap (noted 'OSN+StopGap'). Figure 16 (left plot) presents the results of these experiments. We can notice that both OSN+ (1/2)StopGap and OSN+StopGap perform better than the standard consolidation system (i.e., OSN). In the case of OSN+StopGap, OSN is enhanced with up to 62.5%.

Scalability StopGap complexity depends on the efficiency of the original consolidation algorithm employed by the data center. The worst case complexity corresponds to the use of StopGap as the only consolidation engine. Although this is not its main goal, StopGap can play that role in the absence of a consolidation system. In this case, its complexity is the same as most First Fit

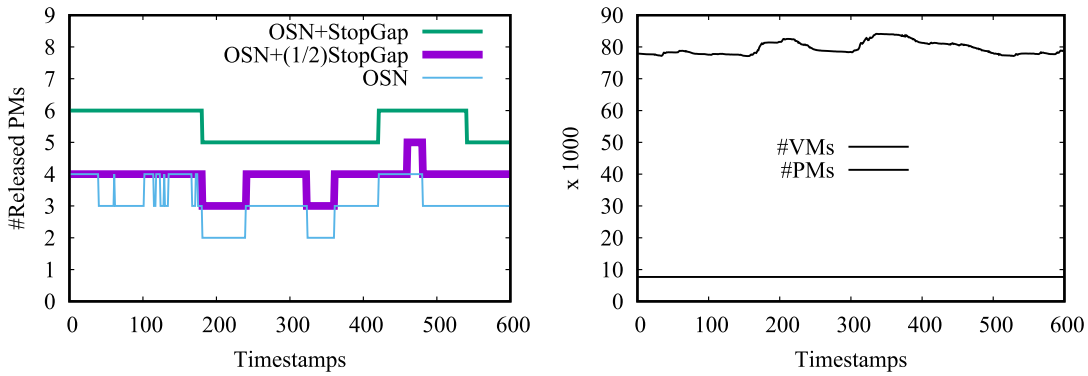

Figure 16. (top) The subset of Google data center traces we used. (bottom) Resource saving on Google traces when our solution is used. OSN, OpenStack Neat; PM, physical machine; VM, virtual machine. [Colour figure can be viewed at wileyonlinelibrary.com]

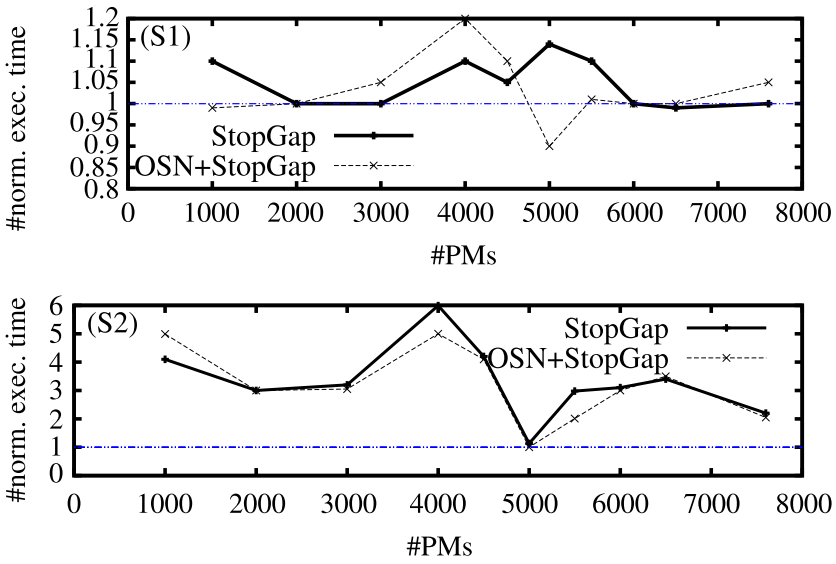

Figure 17. Overhead of our solution. OSN, OpenStack Neat; PM, physical machine. [Colour figure can be viewed at wileyonlinelibrary.com]

Decreasing bin-packing algorithms [19]. The consolidation algorithm used by OSN has the complexity $\mathcal{O}(n \times m)$, where n is the number of PMs and m is the number of VMs to be relocated. We have also relied on Google traces to evaluate and compare both StopGap and OSN scalability. We considered two extreme datacenter states (S1 and S2) which respectively represent the highest and the lowest OSN efficiency. From Figure 16 left, we choose S1 and S2 to respectively be the timestamps 450 and 200. For each situation, we executed three consolidation algorithms (OSN, StopGap, and OSN+StopGap) on different subsets from the original set of PMs. Normalized execution times (against OSN) are plotted in Figure 17. In the most efficient case (S1), we can notice that both StopGap and OSN+StopGap are close to OSN. Conversely, both perform better than OSN when it is not efficient (S2). In this case, StopGap as well as OSN+StopGap does the entire consolidation effort. The minimum value noticed in Figure 17(top) represents another observation: OSN has the highest efficiency when it operates on 5000 PMs.

7. RELATED WORK

Memory footprint improvements. Significant research has been devoted to improve workload consolidation in data centers [36]. Some studies have investigated VM memory footprint reduction to increase VM consolidation ratio. Among these, memory compression and memory over commitment [37, 38] are promising. In the same vein, [39] extends the VM ballooning technique to software for increasing the density of software colocation in the same VM. Xen offers the so called 'stub domain'[¶]. This is a lightweight VM which requires limited memory (about 32 MB) for its execution. *Uncoordinated Policies*. Many research projects focus on improving resource management on client side [40–43]. They aim at improving the workload prediction and the allocation of VMs for replication. On the provider side, research mainly focuses on (i) size of resource slices, that is, provided VM size; or (ii) VM placement, that is, VM allocation and migration across physical servers to improve infrastructure utilization ratio. Various algorithms are proposed to solve the VM packing problem [22, 23]. They take into account various factors such as real resource usage, VM loads, etc. However, in a dataceter, the resource demands of a VM are not fixed. Thus, several authors propose heuristics which address the dynamicity of this problem. Beloglazov *et al.* [44] propose an algorithm which take consolidation decisions based on a minimum and a maximum PM utilization

[¶]http://wiki.xen.org/wiki/StubDom.

threshold. Because live VM migration is a costly operation, Murtazaev *et al.* propose Sercon [45], a consolidation algorithm that minimize not only the number of active PMs but also the number of VM migrations. Further, the state-of-the-art algorithms are leveraged in order to build dynamic consolidation systems. For example, Snooze [46] is an open-source consolidation system build on top of Sercon. Snooze implements a decentralized resource management on three layers: local controllers on each PM, group managers which survey a set of local controllers and a group leader among the group managers. However, the previous solutions operate independently either on the client side or the provider side. For this reason, their potential effectiveness may be narrowed.

Cooperative Policies. Authors in [47] describe a model to coordinate different resource management policies from both cloud actors' point of views. The proposed approach allows the customer to specify his resource management constraints, including computing capacity, load thresholds for each host and for each subnet before an allocation of a new VM, etc. The authors also describe a set of affinity rules for imposing VM collocation in the IaaS, which is a form of knowledge sharing. The authors have asserted that this model allows an efficient allocation of services on virtualized resources. This work is a first step in the direction of coordinated policies. Nguyen Van *et al.* [48] describe research works closely-related to ours. The authors propose an autonomic resource management system to deal with the requirements of dynamic VMs provisioning and placement. They take both application level SLA and resource cost into account, and support various application types and workloads. Globally, the authors clearly separate two resource management levels: Local Decision Modules and the Global Decision Module (GDM). The two are respectively similar to our AppManager and IaaSManager. These two decision modules work cooperatively: the Local Decision Module makes requests to the GDM to allocate and deallocate VMs, the GDM may request back changes to allocated virtual machines. The results reported in [48] are only based on simulations.

Christina Delimitrou *et al.* [2] presents Quasar, a non-virtualized cluster management solution which adopts an approach philosophically close to our. It asserts that the customers are not able to correctly estimate the amount of resource needed by their applications to run efficiently. The customers are allowed to express their needs in terms of Quality-of-Service (QoS) constraints, instead of low level resource requirements. The management system will allocate the appropriate amount of resource which ensures the requested QoS. Like our solution, knowledge about applications and their expected QoS is shipped to the cloud management system. This cooperation enables a smarter resource management. Contrary to our solution, Quasar manages non-virtualized clusters and does not address any dynamic consolidation issues.

Elastic workloads. Zhenhua Guo *et al.* [49] proposes a mechanism to split map-reduce tasks for loadbalancing reasons. Because this application type may also be split, it could be included (along with the MTMSA) in the list of suitable applications for our model. Middleboxes represent an important obstacle in the scalability of web applications. In order to address this limitation, Shriram Rajagopalan *et al.* [50] come up with a framework capable of splitting the middlebox VMs (e.g., loadbalancers, firewalls, and protocol accelerators). Consequently, the entry point of an application (i.e., the loadbalancer) may now be distributed over multiple VMs. This work may exempt us from the need of the traditional model because the entry point of an MTMSA application could now be negotiated at the granularity of a tier (TGRNM).

8. LIMITATIONS AND FUTURE WORK

Our solution is effective only with systems, which consolidate VMs based on booked resources. However, there are also systems which take consolidation decisions based on workload resource utilization. In this latter case, the consolidation system try to predict the workload behavior and may deliberately let holes in order to absorb possible forthcoming peaks. On short term, we plan to adapt StopGap to consolidation systems in this latter case. In this respect, a better cooperation between StopGap and the consolidation system is required.

StopGap is effective only for a part of cloud applications (i.e., elastic applications). In this respect, we investigate broader solutions which address the datacenter fragmentation. The cloud computing metaphor perceive a datacenter as a uniform computing facility and not a set of isolated servers.

Thus, we consider that the cloud software stack should be restructured accordingly. On long term, we plan to architecture and develop a distributed virtualization system which allows a VM to use, at the same time, resources provided by multiple servers.

9. CONCLUSION

This paper proposes a way to combine cooperative resource management with elastic VMs. Knowledge about customer's applications (e.g., tier instances) is shared with the IaaS provider so that IaaSManager can better optimize the infrastructure. Based on this shared knowledge, the provider can split or enlarge VMs. Our proposed cooperative IaaS can be considered from two different perspectives: a PaaS extension or a hybrid IaaS-PaaS model. We validated the applicability of our solution through extensive experiments. Relying on Google datacenter traces, we evaluated our solution's benefits in terms of resource saving. It improves OpenStack consolidation engine by about 62.5%, without any additional overhead.

REFERENCES

1. Barroso L. HCloud: warehouse-scale computing: entering the teenage decade. *ISCA Keynote*, San Jose, CA, USA, 2011.
2. Delimitrou C, Kozyrakis C. Quasar: resource-efficient and qos-aware cluster management. *ASPLOS*, Salt Lake City, UT, USA, 2014; 127–144.
3. AbdelSalam H, Maly K, Mukkamala R, Zubair M, Kaminsky D. Towards energy efficient change management in a cloud computing environment. *AIMS*, Enschede, The Netherlands, 2009; 161–166.
4. Cloudify. http://www.cloudify.cc/ [last accessed 12 January 2017].
5. Roboconf. http://roboconf.net/ [last accessed 12 January 2017].
6. AWS Auto Scaling. https://aws.amazon.com/fr/autoscaling/ [last accessed 12 January 2017].
7. *The Autoscaling Application Block*. https://msdn.microsoft.com/en-us/library/hh680892(v=pandp.50).aspx [last accessed 12 January 2017].
8. Clark C, Fraser K, Hand S, Gorm Hansen J, Jul E, Limpach C, Pratt I, Warfield A. Live migration of virtual machines. *NSDI*, Boston, MA, USA, 2005; 273–286.
9. Liu L, Wang H, Liu X, Jin X, He W, Wang Q, Chen Y. Greencloud: a new architecture for green data center. *ICAC-INDST*, Barcelona, Spain, 2009; 29–38.
10. OpenStack Neat. http://openstack-neat.org/, visited on May 2015 [last accessed 12 January 2017].
11. *Distributed Resource Scheduler, Distributed Power Management*. http://www.vmware.com/fr/products/vsphere/enhanced-app-performance.html [last accessed 12 January 2017].
12. OpenNebula. http://opennebula.org/, visited on May 2015.
13. Barroso LA, Holzle U. The case for energy-proportional computing. *IEEE Computer 40* 2007; **12**:33–37.
14. Meisner D, Gold BT, Wenisch TF. The powernap server architecture. *ACM Transaction on Computer Systems* 2011; **29**(1).
15. Elastic Compute Cloud (EC2) Cloud Server and Hosting - AWS. Retrieved March 03, 2016, from https://aws.amazon.com/ec2/ [last accessed 12 January 2017].
16. Eolas. Retrieved March 03, 2016, from http://www.eolas.fr/ [last accessed 12 January 2017].
17. OpenStack. https://www.openstack.org/, visited on May 2015 [last accessed 12 January 2017].
18. SPECvirt_sc2010. https://www.spec.org/virt_sc2010/, visited on May 2015 [last accessed 12 January 2017].
19. Dosa G. The tight bound of first fit decreasing bin-packing algorithm is FFD(I) $\leqslant$ 11/9OPT(I) + 6/9. *ESCAPE*, Hangzhou, China, 2007; 1–11.
20. Hamilton J. Cooperative expendable micro-slice servers (CEMS): low cost, low power servers for internet-scale services. *CIDR*, Asilomar, California, 2009.
21. Microsoft's Top 10 Business Practices for Environmentally Sustainable Data Centers. http://www.microsoft.com/environment/news-and-resources/datacenter-best-practices.aspx [last accessed 12 January 2017].
22. Bobroff N, Kochut A, Beaty K. Dynamic placement of virtual machines for managing SLA violations. *IM*, Munich, Germany, 2007; 119–128.
23. Nakada H, Hirofuchi T, Ogawa H, Itoh S. Toward virtual machine packing optimization based on genetic algorithm. *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, Salamanca, Spain, 2009; 651–654.
24. Amoeba. http://www.cs.vu.nl/pub/amoeba/amoeba.html, visited on May 2015 [last accessed 12 January 2017].
25. CloudSuite. http://cloudsuite.ch/ [last accessed 12 January 2017].
26. SPECweb2005. https://www.spec.org/web2005/, visited on May 2015 [last accessed 12 January 2017].
27. SPECjAppServer2004. https://www.spec.org/jAppServer2004/, visited on May 2015 [last accessed 12 January 2017].
28. SPECmail2008. https://www.spec.org/mail2008/, visited on May 2015 [last accessed 12 January 2017].
29. Apache. http://httpd.apache.org/, visited on May 2015 [last accessed 12 January 2017].

30. HaProxy. http://www.haproxy.org/, visited on May 2015 [last accessed 12 January 2017].
31. GlassFish. https://glassfish.java.net/, visited on May 2015 [last accessed 12 January 2017].
32. How To Set Up MySQL Master-Master Replication Retrieved March 03, 2016, from. https://www.digitalocean.com/community/tutorials/how-to-set-up-mysql-master-master-replication [last accessed 12 January 2017].
33. Cyrus. https://cyrusimap.org/, visited on May 2015.
34. Advanced traffic control. https://wiki.archlinux.org/index.php/Advanced_traffic_control, visited on May 2015 [last accessed 12 January 2017].
35. https://code.google.com/p/googleclusterdata/wiki/ClusterData2011_2, visited on May 2015 [last accessed 12 January 2017].
36. Amit N, Tsafrir D, Schuster A. VSwapper: a memory swapper for virtualized environments. *ASPLOS*, 2014.
37. Sharma P, Kulkarni P. Singleton: System-wide page deduplication in virtual environments. *HPDC*, Delft, The Netherlands, 2012; 15–26.
38. Barker S, Wood T, Shenoy P, Sitaraman R. An empirical study of memory sharing in virtual machines. *USENIX ATC*, Boston, MA, 2012; p25.
39. Salomie T-I, Alonso G, Roscoe T, Elphinstone K. Application level ballooning for efficient server consolidation. *EuroSys*, Prague, Czech Republic, 2013; 337–350.
40. Quiroz A, Kim H, Parashar M, Gnanasambandam N, Sharma N. Towards autonomic workload provisioning for enterprise grids and clouds. *GRID*, Banff, AB, Canada, 2009; 50–57.
41. Chaisiri S, Lee B-S, Niyato D. Optimal virtual machine placement across multiple cloud providers. *APSCC*, Kuala Lumpur, Malaysia, 2008; 103–110.
42. Jayasinghe D, Pu C, Eilam T, Steinder M, Whalley I, Snible E. Improving performance and availability of services hosted on iaas clouds with structural constraint-aware virtual machine placement. *SCC*, Washington DC, USA, 2011; 72–79.
43. Farahnakian F, Pahikkala T, Liljeberg P, Plosila J, Tenhunen H. Utilization prediction aware VM consolidation approach for green cloud computing. *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, New York, USA, 2015; 381–388.
44. Beloglazov A, Abawajy J, Buyya R. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems* 2011:755–768.
45. Murtazaev A, Oh S. Sercon: server consolidation algorithm using live migration of virtual machines for green computing. *Iete Technical Review* 2011; **28**(3):212–231.
46. Feller E, Rilling L, Morin C. Snooze: a scalable and autonomic virtual machine management framework for private clouds. *12th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid 2012)*, Ottawa, Canada, 2012; 482–489.
47. Konstanteli K, Cucinotta T, Psychas K, Varvarigou T. Admission control for elastic cloud services. *CLOUD*, Honolulu, Hawaii, USA, 2012; 41–48.
48. Van HN, Tran FD. Autonomic virtual resource management for service hosting platforms. *CLOUD*, Bangalore, India, 2009; 1–8.
49. Guo Z, Pierce M, Fox G, Zhou M. Automatic task reorganization in mapreduce. *CLUSTER 2011*, Austin, TX, Sep. 2011; 335–343.
50. Rajagopalan S, Williams D, Jamjoom H, Warfield A. Split/Merge: system support for elastic execution in virtual middleboxes. *NSDI*, Berkeley, CA, USA, 2013; 227–240.