Toward Informed Resource Management in the Cloud

Giang Son Tran, Laurent Broto, Daniel Hagimont

Toward Informed Resource Management in the Cloud

Giang Son Tran, Laurent Broto, Daniel Hagimont
ENSEEIHT - University of Toulouse, France
Email: {first.last}@enseeiht.fr

***Abstract*—More and more companies are externalizing their computing infrastructures to the cloud to reduce the increasing maintenance cost of computing environments. Minimizing the amount of hardware resource and power consumption in use is one of the main services that such a cloud infrastructure must ensure. This objective can be done either by the customer at the application level (by dynamically sizing the application based on the workload), or by the provider at the virtualization level (by consolidating virtual machines based on the infrastructure's utilization rate). Many research works investigate resource management policies separately. In this paper, we show that the different strategies for cloud resource management, including server consolidation only, dynamic application sizing only, both policies at the same time, do not fully bring benefits to the cloud actors when being implemented without cooperation. Finally, we propose and evaluate a cooperative model to combine the efficiency from these strategies in reducing power consumption and keeping application's Quality of Service.**

I. Introduction

As computer systems continue to evolve, companies are externalizing their computing infrastructures to another type of companies called *the provider*. The former is called *the customer*. This movement is to reduce the increasing maintenance cost of computing environments. This paper considers the IaaS (Infrastructure-as-a-Service) model: the provider gives their infrastructure's logical resource in the form of Virtual Machines (VMs), with virtual CPU power, cores, memory, storage, network interface, etc., to the customer.

In this context, resource management is one of the main services that both providers and customers must ensure. The minimization of resources, both in CPU power and memory usage, brings benefits for both actors. The customer only pays for the amount of the used resource (pay-as-you-go). The provider can lower his power consumption for their physical devices (machines and cooling systems) by suspending or turning off or the unused physical machines (PMs).

Various resource management policies in the cloud are investigated: at the customer level [1], or at the provider level [2], [3]. The need of a complement for these resource management strategies raises when being deeply analyzed with specific scenarios, as will be shown in this paper. Our main contributions are (1) briefly describe, evaluate and point out the drawbacks of the existing resource management policies with a synthesized workload; (2) show that resource management at these two levels are complementary and should be coordinated; and (3) propose and evaluate a cooperative resource management policy (fig. 1).

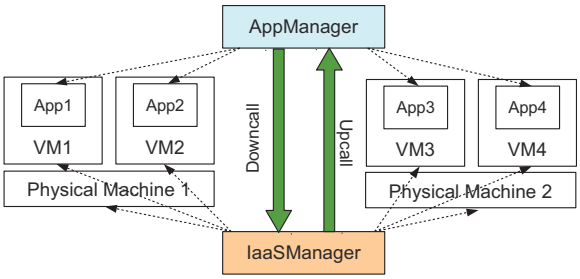

Fig. 1: Cooperative Resource Management Policy with Cooperative Calls

The rest of the article is organized as follows. Section II details the resource management policies and motivates our work. Section III presents our cooperative resource management policy between the two layers. We evaluate and compare the effectiveness of the policies in section IV. After highlighting various related works in section V, we conclude and present our future work in section VI.

II. Motivation

This section motivates our work by describing and pinpointing the drawbacks of the various cloud management policies being investigated in the research community, including: Server Consolidation Only; Dynamic Application Sizing Only; both levels, but working independently.

Server Consolidation Only (SCO). Taking into account the objective of minimizing hardware resource from the provider, this policy is relatively straightforward: pack the deployed VMs into as few PMs as possible using VM migration. In this scenario, only does the provider implement their autonomic resource management system. The customer application is provisioned with a static tier allocation (i.e. with a fixed number of tier instances). In case of overload, the IaaS manager can either migrate the most loaded VM out of the most loaded node (so that this node becomes less load), or migrate other VMs to other free nodes (so that the most loaded VM has more power). The IaaS resize resource pool by adding more PMs, if needed. In contrast, when allocated VMs are under load in runtime, the management policy at the IaaS level will make migration decision to pack the idle VMs into as few PMs as possible. Freed PMs are then suspended or turned off for reducing overall power consumption.

This policy shows some merits in minimizing resource usage, and therefore, energy waste. However, multiple VMs

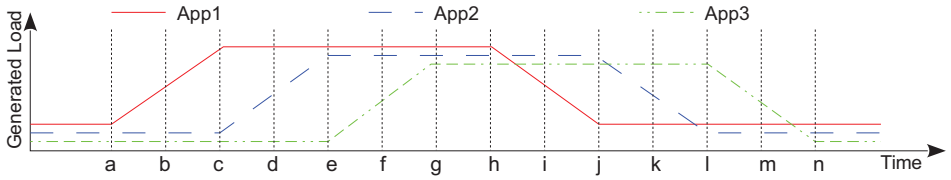

Fig. 2: Synthesized workload used in all scenarios

of the same application tier may share the same PM. This placement produces two types of performance overhead: (1) balancer overhead - each request to the application must be passed through the balancer; and (2) hypervisor overhead - the hypervisor has to switch CPU resources among many VMs, generating overhead. These overheads can be reduced by using less VM instances with Dynamic Application Sizing.

Dynamic Application Sizing Only (DASO). This approach is based on the dynamic allocation and deallocation of the application instances. Initially, all applications are deployed with a minimum number of instances. Each instance is deployed and launched in a separated VM. During runtime, tier loads are captured by monitoring probes in the VMs, and gathered by the autonomic application manager. It, in turns, based on current tier loads, requests to add or remove the VMs accordingly.

This behavior ensures the minimal number of the instances of the application, and therefore reduces performance overhead. However, this policy may create some resource holes: multiple VMs of the same tier may spread on different PMs, and leave available resources on each PM unused. These holes can be filled by migrating the VM to ensure server consolidation, free them for turning off and benefit in energy saving.

Both Levels, Independent (BLI). In this scenario, both the provider and the customer implement their resource management policies (dynamic application sizing and server consolidation) independently, to eliminate each other's drawback. In other word, this complementarity attempts to improve both real resource usage (to reduce energy waste) and application performance (by reducing overhead) in the hosting centers. The dynamic application sizing policy at the customer level ensures that all allocated VMs' usage are optimized: idle or unused VMs are deallocated automatically. Therefore the migration policy is based on the capacity of each VM.

This combination brings significant benefit to both actors. First, there are fewer migrations than SCO, because the migration check is only performed when there is a VM freed. Second, the number of application instances is still minimized, because the application manager deploys its instance on-demand, same as DASO. Finally, more PMs can be freed than DASO, because the IaaS manager optimizes its VM placement with migration, results in reducing power consumption.

However, this placement still has the same problem with SCO: the possibility of having multiple VMs of the same application tier on a PM. This is not optimized for performance with VM overhead and balancer overhead. This problem comes from the fact that the application manager is not aware of its VM location, and that the IaaS manager is not aware of the application tiers.

III. Both Levels, Cooperative (BLC) Resource Management

To overcome the drawbacks of DASO and SCO being implemented without coordination, we propose a cooperative resource management policy. The key difference in this policy, compared with the above policies, is to gather application instances into groups, and then manage groups with quotas instead of VMs. These quotas can be dynamically changed in runtime. This group notion provides the application architecture to the IaaS manager, thus simplifies the VM management. The two layers communicate with each other through cooperation calls (fig. 1). A call from application layer to the IaaS layer is a Downcall. The call in the other direction is an Upcall.

The customer's application manager monitors its tier load, and based on the actual runtime situation, either (1) overload: requests a group quota increase; or (2) under load: requests a group quota reduction. According to the request to modify a group quota Δq, the IaaS manager can:

Add quota ($\Delta q > 0$) for an existing VM: $q_{vm} = q_{vm} + \Delta q$. This is the case when this VM has $0 < \Delta q + q_{vm} < 100$ and its host is free enough (in terms of remaining quota). In case of not having enough free quota in the host, the IaaS manager allocates a new VM, VM_k, and informs the application manager with an Upcall about VM_k to deploy an application instance on it.

Reduce quota ($\Delta q < 0$) for an existing VM: $q_{vm} = q_{vm} - |q|$, only possible when $q_{vm} < |q|$. If no VM satisfies this constrain, the IaaS manager reduces quota of several VMs and/or stop a running VM. The notification about this tier reconfiguration will also be sent to the application manager for tier reconfiguration.

The IaaS checks for optimizations after changing quota. We identified several situations for tier optimization from our experiment: migrations of VMs for freeing a PM, VM resize is not required; or merge of collocated VM from the same group to reduce overhead; or split of a big VM into smaller VMs then migrations of these small VMs to free PMs.

IV. Evaluations

The objective of this section is to prove the effectiveness in minimizing resource usage and maximizing performance of the cooperation strategy. We show the behavior of each strategy with a hypothesized workload (fig. 2) to better compare the benefits and the drawbacks of each policy.

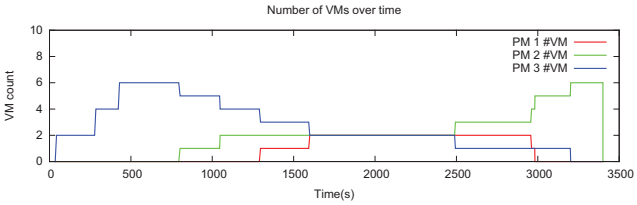

Fig. 3: SCO: VM Allocation

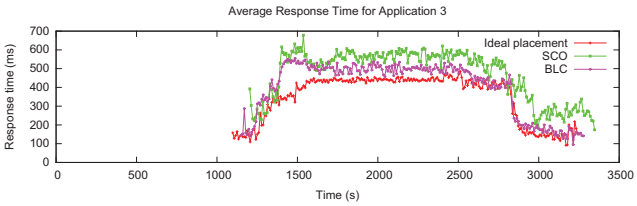

Fig. 4: Response time of Application 3

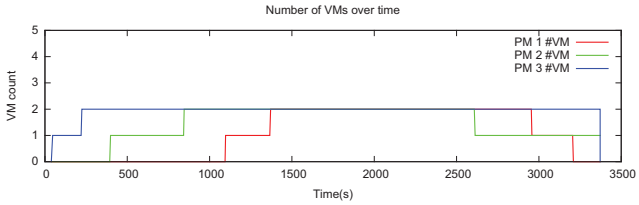

Fig. 5: DASO: VM Allocation

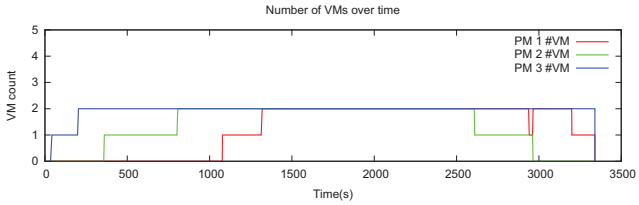

Fig. 6: BLI: VM Allocation

Given a cap value $0 < c_{k,i} \leq n_c$ (number of cores per PMs) in each duration $t_{k,i}$ (in seconds) for a VM VM_k, we define the **VM occupation** in our experiments as $\omega_k = \sum_{i=1}^{n} c_{k,i} \times t_{k,i}$. From this definition, we have a VM occupation of each application App_j as $\Omega_j = \sum_{k=1}^{n} \omega_k$. This metric can be used to evaluate the effectiveness of each resource management policy toward minimizing the booked resource for the customers.

A. Experimental Setup

Our experiments were performed in a private cluster of 7 nodes, equipped with an Intel Core 2 Duo 2.66GHz (2 cores), 4GB RAM and running Debian Squeeze with Xen 4.1.5 as the hypervisor. The RUBiS application is considered as the customer's multi-tier applications. We consider only MySQL tier in our experiments to simplify the management policies.

B. Evaluating performance overhead in SCO

Fig. 3 shows the VM placement for SCO. Initially two MySQL server instances are deployed for each application (provisioned). Before the experiment, all VMs for idle MySQL instances are packed into PM3 (500th-700th second). There are several VM migrations (800th, 1050th, 2900th... seconds) when the application loads rise and drop. From 1600th second to 2450th second, each PM has two VMs of the same application running at their maximum workload.

Fig. 4 compares the average response time of the third RUBiS application with different scenarios, including an ideal VM placement: only one big VM of each application is placed on each PM. From the figure, the response time of SCO during full load is approximately 10%-15% higher than the one in the ideal VM placement, because of the balancer and hypervisor overhead. This overhead shows the trade off between minimizing PM usage and maximizing performance.

C. Evaluating power waste in DASO

There is no migration in the DASO policy (fig. 5): all the resource management jobs are handled by the customer application managers. Initially, each application deploys only one instance of MySQL(400th second). MySQL instances are then dynamically added (800th, 1050th, and 1300th second) and removed (2600th, 2950th, and 3250th second) according to the CPU load of this tier, to ensure response time and a minimal number of VMs.

Fig. 5 shows a possibility to consolidate PM2 and PM1 at time 2950th second of the experiment: PM2 and PM1 each have one VM. This placement creates two holes, each on a PM. However, since there is no consolidation strategy implemented in this scenario, these under-used nodes are kept intact until the deallocation of a VM on PM1 at 3250th second. A node in our cluster consumes approximately 60 watts in full-load state, 50 watts in its half-load state and 3 watts in sleep mode. If one VM is migrated from PM1 to PM2, PM2 is full loaded (60W) and PM1 can be suspended (3W). Therefore, 37W is wasted in a 300s duration of this experiment.

D. Evaluating the overhead of two policies in BLI

We use the same generated workload in this experiment to confirm the benefits (in minimizing hardware usage) and drawbacks (in performance overhead) of these two policies when being used at the same time (fig. 6). All three RUBiS applications are started with three MySQL servers (one for each application). Each MySQL VM requires half of a PM, in terms of CPU and memory quota. The initial VM allocation is relatively similar to one in fig. 5. The benefit from triggering a consolidation is shown at 2950th second: a VM from PM2 is moved to PM1, freeing up PM2 for suspension to save energy. Compared to section IV-C, this migration saves 37W from 2950th to 3250th second.

However, BLI still has a higher response time than the ideal VM placement (only one VM for each application on one PM). In our experiment with BLI, response time of the application 3, similar to the one in DASO, is still approximately 10%-12% higher than the ideal scenario in full load (fig. 4). This difference shows the performance drawback for BLI.

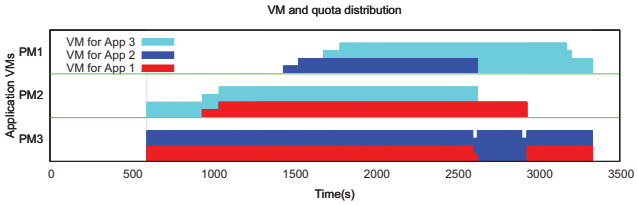

Fig. 7: BLC: VM and Quota Distribution

E. Evaluating two policies at the same time, with cooperation

The cooperation scenario involves two actors at the same time. The cooperation calls are sent in runtime to notify each actor's ongoing situations and actions. The VM and quota allocation of all applications in this experiment are shown in fig. 7: each application uses Downcalls to ask for quota increase during the upramp phase (950th, 1050th, 1450th, 1550th, etc second). Depending on the VM placement and available quota on each PM in the time of those Downcalls, the IaaS either increases quota for an existing tier VM (1050th, 1550th and 1750th second) or allocates a new VM (950th, 1450th and 1650th second). In the later case, the IaaS uses Upcalls to notifies this allocation to the customer application manager, so that a new application tier instance is deployed. Similarly, the possibilities to decrease a tier quota are handled in the downramp of the workload. According to the customer application manager's Downcalls, the IaaS then decides to reduce quota for a tier VM (2600th, 2900th and 3150th second) or to remove it (2650th and 2950th second).

Notice that during runtime, with the knowledge provided by the application managers, the IaaS manager proposes to merge small VMs into bigger ones, in attempt to reduce overhead. For example, a cooperative merge happens at 2650th second: the IaaS merges two VMs for application 2 (in PM1 and PM3) into one big VM (in PM3). At the same time, another cooperative merge is executed between two VMs of application 3 (in PM1 and PM2) into a big VM in PM1. Application 3 then benefits with a single instance running in a big VM: its response time reaches to the level of ideal VM placement from 2650th second to the end of the experiment (fig. 4). Additionally, after reducing quota for the VM of application 2 at 2950th second, the IaaS migrated the VM for application 1 from PM2 to PM3 and turned PM2 off.

F. Comparison

Table I summarizes the capabilities of the above policies. The VM occupation values show the total of booked resource, and the total PM utilization times show the hardware resource usage. The higher PM utilization time, the more energy the provider needs to spend. Similarly, the higher value of the booked resource, the more the customer has to pay.

Table I confirms the goals of SCO and DASO: SCO minimizes the amount of PMs being used for the provider (7419s), and DASO minimizes the booked resources for the customer ($\Omega_{DASO} = 2131$). However, each policy brings disadvantages to the other actor when being implemented separately: Ω_{SCO} is very high (almost doubles the amount of the remaining scenarios), and PM utilization of DASO is 12.5% higher than the one of SCO. BLI attempts to reduce the PM utilization of DASO by 288 seconds by migrating a VM from PM2 to PM1. This total reduction of PM usage confirms the benefit of this scenario in section IV-D. Our proposed BLC has slightly higher hardware resource usage than SCO (2.5%, 7611s compared to 7419s), but greatly reduces the booked resource for the customer (53% of SCO). In short, BLC reduces cost to both the customer and the provider.

Value	SCO	DASO	BLI	**BLC**
PM Utilization (s)	7419	8347	8059	**7611**
Average VM Occupation Ω	3902	2131	2136	**2083**

TABLE I: Comparison of the Policies

V. Related Works

Few systems addressed dynamic resource management at both levels. [4] proposed a two-level resource management, but their resource provisioning at the hosting center level was only based on the allocation of additional resource to VMs. [5] proposes a solution to the coordination problem between VMs and the hosted applications when the resource availability has changed. The [5]'s approach uses a hybrid solution (a feedback learning solution combine with a proactive solution) to prevent the reconfiguration of VMs and the applications they run. Most two-level resource management systems did not provide a cooperative strategy for these two levels, and thus, did not achieve optimal energy saving and performance.

VI. Conclusion and Perspectives

This paper shows that cloud resources can be managed at two levels: at the application level and at the IaaS level. Moreover, it shows that resource management strategies at these two levels are complementary, especially when these two levels work cooperatively. We are currently conducting performance evaluations with real workload (monitored in a real hosting center), instead of synthesized workload, to demonstrate the effectiveness of this approach. A longer term perspective of this work will be to consider an optimal algorithm for VM placement and quota management based on workload prediction.

References

[1] H. AbdelSalam, K. Maly, R. Mukkamala, M. Zubair, and D. Kaminsky, "Towards energy efficient change management in a cloud computing environment," in *Scalability of Networks and Services*. Springer, 2009.

[2] L. Liu, H. Wang, X. Liu, X. Jin, W. B. He, Q. B. Wang, and Y. Chen, "Greencloud: a new architecture for green data center," in *Proceedings of the 6th international conference industry session on Autonomic computing and communications industry session*. ACM, 2009.

[3] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall, "Entropy: a consolidation manager for clusters," in *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. ACM, 2009, pp. 41–50.

[4] Y. Song, H. Wang, Y. Li, B. Feng, and Y. Sun, "Multi-tiered on-demand resource scheduling for vm-based data center," in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE Computer Society, 2009, pp. 148–155.

[5] X. Bu, J. Rao, and C.-Z. Xu, "A model-free learning approach for coordinated configuration of virtual machines and appliances," in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*, 2011.