Two-Level Cooperation in Autonomic Cloud Resource Management

Giang Son Tran[a], Alain Tchana[b], Laurent Broto[a], Daniel Hagimont[a]
[a] ENSEEIHT – University of Toulouse, Toulouse, France
Email: {giang.tran, laurent.broto, daniel.hagimont}@enseeiht.fr
[b] University of Joseph Fourier, Grenoble, France
Email: alain.tchana@inria.fr

***Abstract*— Virtualized cloud infrastructures are becoming very popular as they allow separation of hardware and software management. Infrastructure as a Service (IaaS) is the model providing many advantages to both provider and customer. Minimizing the number of resource (and power consumption) in use is one of the main services that such an cloud model 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 at the application level or at the virtualized level. In this paper, we study different strategies for cloud resource management: virtual machine consolidation only, dynamic application sizing only, both policy at the same time (either independent or cooperative). We show that virtual machine consolidation and dynamic application sizing do not fully bring benefits to the cloud provider and customer when being implemented without cooperation. Finally, we propose a cooperative model to improve the efficiency of these strategies, in reducing power consumption and keeping application's Quality of Service.**

***Index Terms*— Cloud computing, cooperative, resource management.**

I. INTRODUCTION

Cloud computing is the current trend of separating hardware and software management, improving the devotion of the *customers* and the *providers*: the customer only needs to manage their applications without the need of hardware maintenance; while the provider is expected to ensure Quality-of-Service to the customer according to their Service Level Agreement. Cloud hosting infrastructures are generally split into 3 categories: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). In this article, we consider a cloud as an IaaS: a virtualized infrastructure managed by the provider. Virtual machines are provided to the external customers to deploy and execute their applications.

In this context, on demand resource management is one of the main services that such an environment must ensure: the allocation of resource as needed and the deallocation when unused. The number of machines is therefore optimized, and energy consumption is reduced.

On demand resource management can be handled either at the customer level (i.e., by the administrator of the deployed application), or at the provider level (i.e., by the administrator of the virtualized infrastructure).

Our main contributions in this paper are:

- We describe various resource management policies, advantages and disadvantages of each with a hypothesized workload.
- We show that resource management at the customer and provider levels are complementary.
- We propose a cooperative resource management policy for these two levels.

We experimented these management policies in virtualized environments with a multiple-tier web application. We implemented an autonomic management system jTune, based on TUNe[1], as the deployment and resource management system.

The rest of the article is organized as follows. Section **Error! Reference source not found.** describes the article context regarding virtualization and cloud computing. Section **Error! Reference source not found.** motivates our work. Section **Error! Reference source not found.** details the resource management policies. Section **Error! Reference source not found.** presents our cooperative resource management policy between the two layers. Section **Error! Reference source not found.** highlights various related works. Finally, we conclude and present our future work in section **Error! Reference source not found.**.

II. CONTEXT

A. Virtualization

Virtualization is a software- and/or hardware-based solution for building and running many operating systems simultaneously on the same bare hardware. Those operating systems are named *guest OS* and their execution environment is called *Virtual Machine(VM)*. The *Virtual Machine Monitor (VMM)* or *hypervisor* represents the virtualizing software responsible of hardware emulation and communication between guest OS and devices. The guests OSes are guaranteed to be isolated from each other, providing better security for the applications than when being deployed unvirtualized on the same physical machine.

With the help of virtual machine migration [2], the provider can move executing virtual machines across different physical machines easily and rapidly, allowing

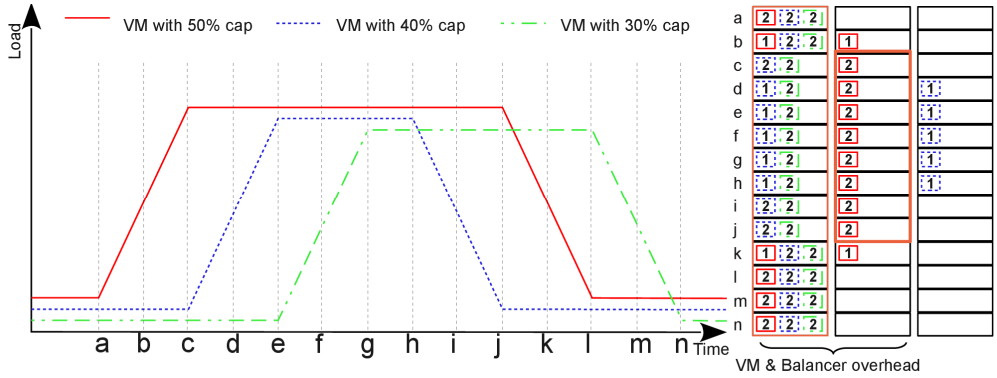

Figure 1: Server Consolidation Only

separation of hardware and software, and consolidating clustered hardware into a single coherent management domain (virtual machines). Therefore, in cloud computing, virtual machines are provided to the customer instead of physical machines. Server utilization is greatly improved with runtime dynamic allocation and deallocation of virtual machines on the physical machines, and thus, reduce power consumption for the provider.

B. *Cloud Computing*

Cloud computing connects the needs of the customer (the application manager) and the services of the provider (the hardware manager). The provider shares the same resource pool to all customers, and provides on demand resource management on it. This strategy brings the benefits for both actors:

- Customer's economy: only needs to focus on application management, leaves the hardware side to the provider, and only pays for the real usage.
- Dynamic capacity of the customer's application: based on the real runtime load, the customer can resize the application (modify the number of virtual machines executing application instances) to handle load peaks or idle states.
- Shared resources: all customers share the same resource pool in the provider's hosting center. Unused machines are switched off, and therefore this strategy provides higher hardware utilization rate and *less energy waste*.

In cloud computing, the customer generally does not have knowledge of the provider's infrastructure, and only has access to the resource in the form of virtual machines.

III. MOTIVATION

Application sizing and server consolidation, with the help of virtual machine migration, have proved their effectiveness in the hosting centers [3][4][5]. However, research works only deducted in these policies separately. Server consolidation with virtual machine migration has the limitation of memory amount of the host: all virtual machines, although being idle, consume memory of the host. Further more, VM overhead increases along with the number of running VMs [6]. Finally, the lack knowledge of the application tier prevents the provider from having an optimal virtual machine placement.

On the other side, application sizing does not fully optimize hardware resources: there is a high possibility that many virtual machines are spread among the physical machines, leaving them unable to free for switching off. These disadvantages show some limitations which can be improved with a two level management, as analyzed in the below sections.

IV. MANAGEMENT POLICIES

This section describes the various cloud management policies being investigated in the research community, including: server consolidation only; dynamic application sizing only; both policies, but working independently. We also describe our experiments for each scenario, and pinpoint the drawbacks of each policy when being used in a hosting center.

We use a typical web application with the Apache – PHP – MySQL stack as the customer application. Fig. 1 (left) shows the synthesized workload we generated to the 3 different web applications. Before time (a), all of the applications are idle (almost no request is generated). Start from time (a) to time (c), application 1 load is increased. Application 2 load is increased from time (c) to (e), followed by increase load of the application 3, from time (e) to (g). These loads are then decreased with the following order: application 2 (time (h)-(j)), application 1 (time (j)-(l)), then application 3 (time (l)-(n)). This synthesized workload is used in all experiments throughout the paper to better compare the benefits and the drawbacks of each policy.

Our experiments were performed in a private cluster of 7 Nodes, equipped with an Intel Core 2 Duo 2.66GHz in single processor mode, 4GB RAM, and Debian Squeeze. All of the Nodes are connected with a 100Mbps Ethernet. The VM disk images are stored on a NFS server so that VM migration can be performed between the nodes. We use Xen 4.1.4 as the hypervisor in our experiments.

A. *Server Consolidation Only*

Taking into account the objective of minimizing hardware resource of the provider, this policy is straightforward: pack the deployed virtual machines into as few physical machines as possible. However, the number of VMs in one physical machine is limited by the

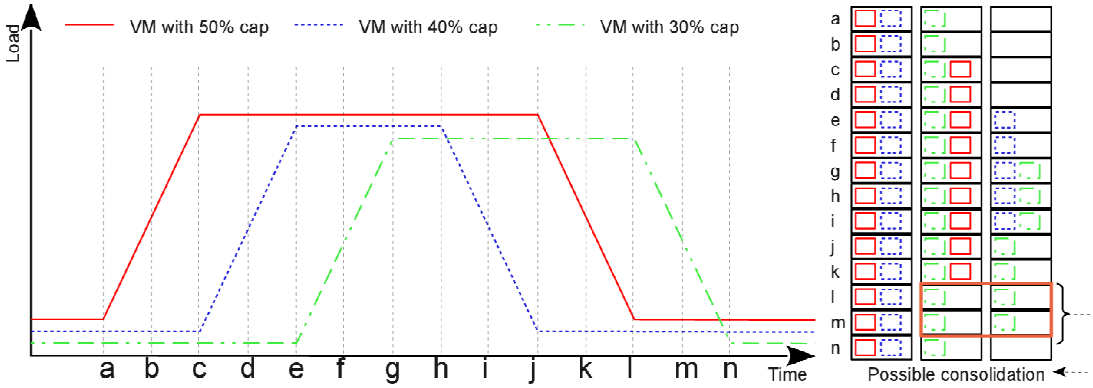

Figure 2: Dynamic Application Sizing Only

memory amount of the host. This policy is made possible with the support of virtual machine migration.

In this scenario, only the provider level is implemented with autonomic management. The customer application is provisioned with a static tier allocation (i.e. with a fixed number of tier instances). The management policy at the provider level takes into consideration each virtual machine CPU load, and makes migration decision: 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 provide more power for the most loaded VM).

Fig. 1 (right) shows the VM allocation on each physical machine according to the generated workload. Each node is equipped with a monitoring probe, periodically reporting CPU load of all VMs on it. These results show the reaction of the IaaS manager toward generated workload to the applications based on the actual VM CPU load. At time (a), all 6 VMs are packed into Node 1. The IaaS manager decides to migrate a VM of the application 1 (red) from Node 1 to Node 2. Gradually, when the loads of all applications increase, the VMs are distributed among physical Nodes (time (b)-(k)).

This policy shows some merits in minimizing resource usage (and therefore, energy waste). However, it still produces several types of performance overhead because of running multiple VMs of the same tier simultaneously:

- Live VM migration overhead: migrating VMs between physical hosts is costly.
- Balancer overhead: each request to the application must be passed through the balancer.
- Hypervisor overhead: The hypervisor has to switch CPU resources among many VMs, generating overheads.

These overheads can be reduced by using less VM instances with dynamic application sizing. This method will be described in the next section.

B. *Dynamic Application Sizing Only*

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 manager. It, in turns, based on current tier load, requests to add or remove the VMs accordingly. Dynamic application sizing is quite generic as it can be applied to any multiple-tier applications.

Fig. 2 (right) shows the VM allocation on physical machines. As the load increases from time (c) to time (g), the application manager gradually deploys and launches more application instances on Node 2 and 3. When the load decreases, the VMs containing these instances are also removed from the hosts.

This behavior ensures the minimal number of the instances of the application, and therefore reduced performance overhead over the previous policy (Server consolidation only). However, this policy raises some possible optimizations at time (l) and (m): two VMs of the same tier (application 3) are running on the two different physical machines. This placement can be improved by migrating the VM from Node 3 to Node 2, and free Node 3 for turning off, benefiting in energy saving.

This drawback can be solved with the combination of the two above policies: the IaaS manager ensures server consolidation with virtual machine migration, and the application manager optimizes its tier allocation. This combined policy is described in the next section.

C. *Both Levels, Independent*

In this scenario, both the customer and the provider implement their resource management policy 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 (the idle or unused VMs are deallocated automatically). Thus, it isn't necessary for the IaaS manager to migrate its VMs based on the CPU load. Instead, the migration policy is based on the capacity of each VM.

Fig. 3 shows the generated workload and the VM placement among the physical machines. Similar to the scenario in IV.B, one instance of each application is deployed initially. When the workload increases, the application manager gradually deploys and launches

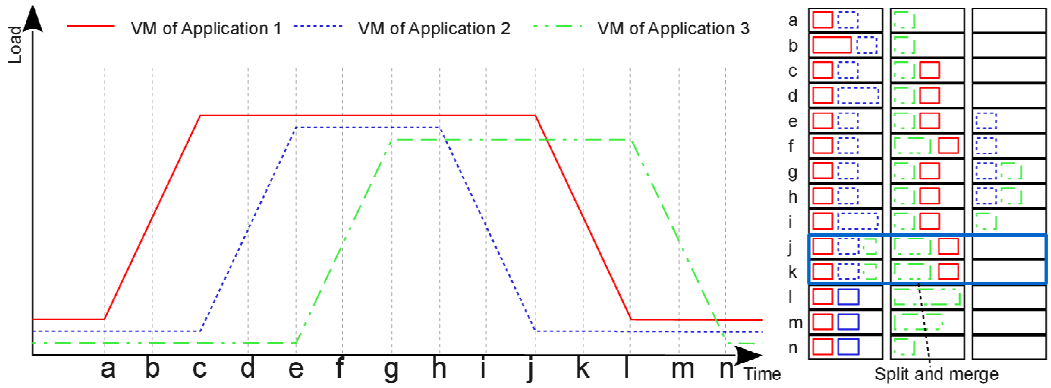

Figure 4: Both Policies, Independent

more application instances on Nodes 2 and 3 to ensure application response time. The IaaS manager's migration check is activated upon receiving a VM removal request from the customer level. In this scenario, it only performs a migration after a VM of the application 1 is removed from Node 2 at time (k).

Comparing these results to those we obtained with management at application level only (IV.B), and IaaS level only (IV.A), we have significant improvements. First, there are fewer migration (one time, at time (k), compared to 4 in (IV.A)). Second, the number of application instances is still minimized, same as (IV.B). Finally, Node 3 can be freed from time (l), when the IaaS manager optimizes its VM placement. This migration, in turn, helps to reduce power consumption of Node 3 when compared with (IV.B).

However, comparing this result with (IV.A), we still have the same problem: the possibility of having multiple VMs of the same application tier on a physical machine (time (l), (m)). This is not optimized for performance with VM overhead and balancer overhead, as previously discussed in (IV.A). 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 tier. The next section describes our proposal to overcome this problem, in order to minimize both power consumption and performance overhead.

V. Cooperative Resource Management

The key difference in this policy, compared with the above policies, is to gather application instances into groups, and 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. A call from application layer to the IaaS layer is a *Downcall*. The call in the other direction is an *Upcall*. In our experiment, these cooperation calls are made possible thanks to Java RMI. Fig. 5 shows the architecture of the cooperation calls between layers.

The customer's application manager monitors its tier load, and based on the actual runtime situation, either:

- Overload: requests a group quota increase, or
- Under load: requests a group quota reduction

According to the request to modify a group quota Δ_q, either it's an addition ($\Delta_q > 0$), or it's a subtraction ($\Delta_q < 0$) the IaaS manager can:

- Add quota ($\Delta_q > 0$) for an existing Vx: $q_{vm} = q_{vm} + \Delta_q$. 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 virtual machine, VM_k, and notifies the application manager about VM_k to deploy an application instance on it.
- Reduce quota ($\Delta_q < 0$) for an existing VM: $q_{vm} = q_{vm} - |\Delta_q|$, only possible when $q_{vm} < |\Delta_q|$. 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.

After every quota change, the IaaS manager always checks for possibility of server consolidation. In our experiment with the synthesized workload (Fig. 4), we

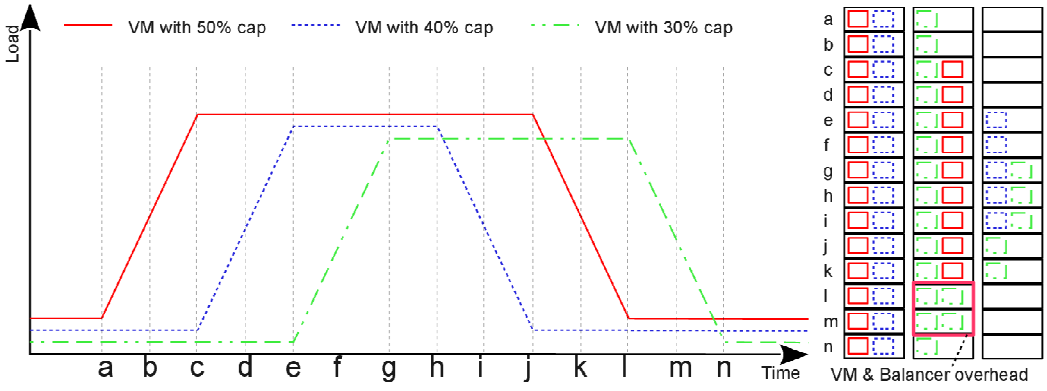

Figure 3: Both Policies, Cooperative

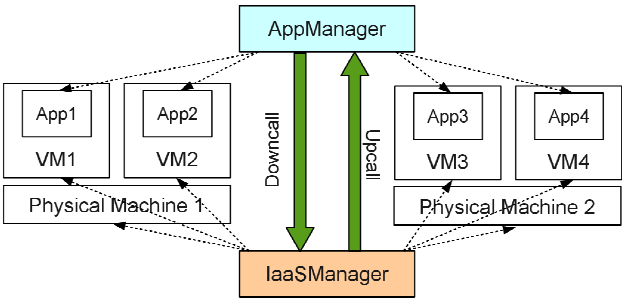

Figure 5: Cooperative calls

identified several possible consolidation situations:

- Only migrations of VMs for a possibility of freeing a physical machine.
- Merges of collocated VM from the same group: time (l), when the IaaS manager merges two small VMs from Node 1 and 2 into one VM with a bigger quota on Node 2, reducing overheads.
- Splits of a VM to smaller VMs, in attempts to free a Node: time (j).

As can be seen from Fig. 4, these two levels, when implemented to work cooperatively, effectively optimize hardware resources (Node 3 is only used in 5 time slots, from time (e) to (i), similar to 5 time slots in (IV.A)), as well as minimize the performance overhead due to live migrations, hypervisor and balancer (similar to IV.B). In summary, the cooperative resource management policy combines all possible advantages, and provide a greater benefit for the customer (less used VMs, and therefore, less cost) as well as for the provider (less hardware resources, and thus, less power consumption). However, it requires both the provider and the customer to have a common API and protocol for the communication of each party's manager.

VI. Related Works

Many research works investigated dynamic resource allocation in the hosting center environments. [7] presents a dynamic allocation architecture for a hosting center based on an autonomic computing system and a load balancer. Similarly, [8] proposed many strategies: jobs distribution to a pool of VMs in a cloud infrastructure, based on dynamic VM allocation/deallocation: new VMs are deployed/undeployed when being overloaded and idle, respectively. Regarding our classification, these solutions are only customer-level strategies.

Second category includes systems which implement resource management at the IaaS level. Consolidation systems such as GreenCloud [9] or [10] aim at saving resource in a hosting center using solely VM migration.

Finally, only few systems addressed dynamic resource management at both levels (application and IaaS). [11] 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. 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.

VII. Conclusion and Perspective

This paper describes different scenarios which consist in ensuring dynamic resource allocation for a cloud in a hosting center. It shows that resources can be managed at two levels: at the level of the application layer and at the IaaS level. Moreover, it shows that resource managements 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 work load prediction.

References

[1] L. Broto, D. Hagimont, P. Stolf, N. Depalma, S. Temate, "Autonomic management policy specification in Tune", in *Proceedings of the 2008 ACM symposium on Applied computing*, Brazil, 2008.

[2] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, A. Warfield, "Live migration of virtual machines", in *Proceedings of the conference on Symposium on Networked Systems Design & Implementation* - Volume 2, 2005.

[3] 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*, USA, 2011.

[4] 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*, China, 2009.

[5] Rightscale web site, in http://www.rightscale.com, visited in December, 2012.

[6] A. Tchana, S. Temate, L. Broto, and D. Hagimont, "Autonomic resource allocation in a J2EE cluster", in *Utility and Cloud Computing*, India, December 2010.

[7] H. S. AbdelSalam, K. Maly, R. Mukkamala, M. Zubair, and D. Kaminsky, "Towards energy efficient change management in a cloud computing environment." in *AIMS, Lecture Notes in Computer Science*, vol. 5637, Springer, 2009.

[8] S. Genaud and J. Gossa, "Cost-wait Trade-offs in Client-side Resource Provisioning with Elastic Clouds," in *IEEE CLOUD*, USA Washington DC, 2011.

[9] L. Liu, H. Wang, X, Liu, X. Jin, en B. He, Q. B. Wang, and Y. Chen, "GreenCloud: a new architecture for green data center," in *International conference industry session on Autonomic computing and communications industry session (ICAC-INDST)*, Spain 2009.

[10] Pablo Graubner, Matthias Schmidt, and Bernd Freisleben, "Energy-Efficient Management of Virtual Machines in Eucalyptus," in *IEEE CLOUD*, USA, 2011.

[11] Y. Song, H. Wang, Y. Li, B. Feng, and Y. Sun, "Multi-tiered on-demand resource scheduling for vm-based data center," in *IEEE/ACM International Symposium on Cluster Computing and the Grid*, May 2007.