Dynamic Scalability of a Consolidation Service

Ahmed El Rheddane[*], Noël de Palma[*], Fabienne Boyer[*], Frédéric Dumont[†], Jean-Marc Menaud[†], Alain Tchana[*]
[*]*LIG/UJF — Grenoble, France*
{*ahmed.elrheddane, noel.depalma, fabienne.boyer, alain.tchana*}*@imag.fr*
[†]*Mines de Nantes, Inria, Lina — Nantes, France*
{*frederic.dumont, jean-marc.menaud*}*@emn.fr*

***Abstract*—In the coming years, cloud environments will increasingly face energy saving issues. While consolidating the virtual machines running in a cloud is a well-accepted solution to reduce the energy consumption, ensuring the scalability of the consolidation service remains a challenging issue. In this paper, we propose an elastic consolidation service that scales according to the dynamic needs of the cloud environment. Our proposition is based on (i) virtualizing the consolidation manager, (ii) partitioning the consolidation work and (iii) regulating the consolidation scalability through an autonomic control loop. Our proposition has been tested and validated through several experiments.**

***Keywords*-cloud computing; IaaS; elasticity; virtualization; consolidation;**

I. Introduction

In order to reduce the maintenance cost of computing environments, companies are increasingly externalizing their computing infrastructures which are therefore managed in a mutualized way in large scale datacenters. Cloud computing is a paradigm which follows this direction using virtualization. The motivations for cloud computing platforms are different with regard to the users or the providers: users want a simple, efficient, reliable and cheap access to the infrastructure which hosts their applications or data, whereas providers want to meet the requirements of their customers while lowering the maintenance cost of the infrastructure.

With the development of cloud infrastructures, the energy consumption has become an important concern. Many studies tried to evaluate the energy used by datacenters. According to the U.S. Environmental Protection Agency (EPA), server farms were consuming 1.2% of US electricity in 2005, and 1.5% in 2006. It amounts for a total electricity cost of about $4.5B (more than the electricity consumed by the nation's televisions). Electricity used by datacenters has doubled between 2000 and 2006 and the projections say it could have doubled by now.

Consolidation is a well known solution for energy saving in the context of virtualized systems. It relies on virtual machine migration in order to transparently and periodically relocate any application in a datacenter. The placement policy takes into account the CPU and memory usages, in order to concentrate the VMs on fewer nodes of the datacenter, thus allowing unused nodes to be shut down, or put into low-energy mode.

However due to the scale of the infrastructure, consolidation services have to be scalable. The workload of the consolidation service has a periodically flat and peak shape and roughly depends on the number of physical and virtual machines in the environment, which represent the system's configuration. One solution would be to rely on replication and static provisioning to scale the consolidation service; this naturally leads to energy waste because the consolidation service will be provisioned to deal with the worst case working set.

The contribution of this paper is to provide an elastic consolidation service that dynamically provision just enough resources for itself to achieve a consolidation in a given time frame, i.e., the desired timeout. This elastic service is a control loop that is able to dynamically provision consolidation managers. By doing this, we promote and demonstrate the use of elasticity at the level of a IaaS management layer in the case of a consolidation service. Our solution is based on:

- the virtualization of the consolidation manager;
- a real time estimation of the working set;
- a simple but efficient probabilistic dynamic partitioning of the configuration.

While this work is based on a specific consolidation manager, it can be easily adapted to fit any other consolidation solution as it considers the consolidation computation as a black box, and enhances it with an elasticity layer. Performance evaluation shows that this solution, albeit simple, provides very good results.

The rest of this paper is organized as follows: Section II presents the consolidation service, Section III explains our design principles, Section IV describes our performance evaluation, Section V positions our solution in the context of energy management and finally, Section VI concludes this paper.

II. Consolidation

A. Definition

One of the top concerns when dealing with large datacenters, which is the typical case of cloud infrastructures, is energy efficiency. This aspect of the so-called green computing, or environment-friendly computing, is already

978-0-7695-5028-2/13 $26.00 © 2013 IEEE
DOI 10.1109/CLOUD.2013.62

promoted by cloud computing, since gathering the computational resources in one big centralized infrastructure, in an optimal place with regard to cooling for instance, can considerably decrease the energy consumption. However, since cloud computing often relies on a virtualized infrastructure, further energy efficiency can be achieved through "consolidation", which is basically minimizing the number of running physical machines by optimally placing the virtual machines of the datacenter. Since the cloud ecosystem is highly dynamic and the virtual machines' needs in terms of resources (memory and CPU) vary, consolidation has to be done periodically.

Our work is based on Entropy [1] which is an open-source consolidation manager. Based on a given infrastructure's configuration (the mapping of the virtual machines on the physical nodes, the virtual machines' current memory and CPU usage and the nodes' total CPU and memory capacity), it would try to minimize the cloud's energy consumption and computes a new mapping, i.e., new configuration, that would minimize the number of used nodes while guaranteeing enough resources for the virtual machines to run normally, and then reconfigure the system accordingly by issuing the appropriate migration commands. In its current version, Entropy can either be executed periodically, as the system's configuration is highly dynamic in the context of cloud computing, particularly regarding the number and placement of virtual machines, or on an event-driven basis, if we want to insure a greater reactivity. Figure 1 depicts Entropy's reconfiguration loop.

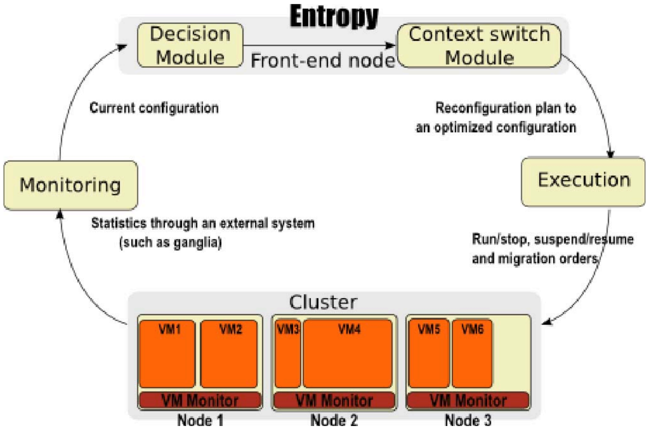

Figure 1. Entropy's reconfiguration loop [1]

Computing a new configuration is a linear optimization problem; to solve it, Entropy uses ChocoSolver [4]. The optimization problem does not only compute the minimal number of nodes needed to run the current virtual machines, but computes a consolidation plan as a whole, which also minimizes the cost of reconfigurations leading to such an optimal configuration. The cost of a reconfiguration roughly depends on the number of migration operations needed to go from a given configuration to the newly computed one. Entropy also supports live migration to execute the computed reconfiguration plan, which makes the execution fairly faster but requires the virtual machines' disks to be stored in a distributed file system such as NFS. More details on how Entropy works can be found in [1] or [2].

While Entropy is undeniably a useful tool, the fact that it relies on solving a linear optimization problem makes it potentially unable to compute an acceptable reconfiguration for realistic configurations' sizes within reasonable timeouts, hence the need to enhance its scalability. Since the cloud's configuration and scalability vary dynamically, we propose to scale Entropy dynamically as well.

III. Design Principle

The scaling of Entropy, as a service provided by the IaaS infrastructure, is driven by the dynamic evolution of both the size and population of the datacenter, which corresponds to the number of virtual machines and physical nodes. An important point is that we do not take into consideration scalability requirements with regard to the clients' requests load.

The main principle behind our solution is to create just enough Entropy instances so as to adapt to the configuration to be processed. To do so, we use (i) a classical autonomic computing [5] design, (ii) node partitioning for scalability purposes and (iii) virtualization in order to enhance the flexibility of managing the Entropy instances and insure elasticity.

A. *Autonomic Loop*

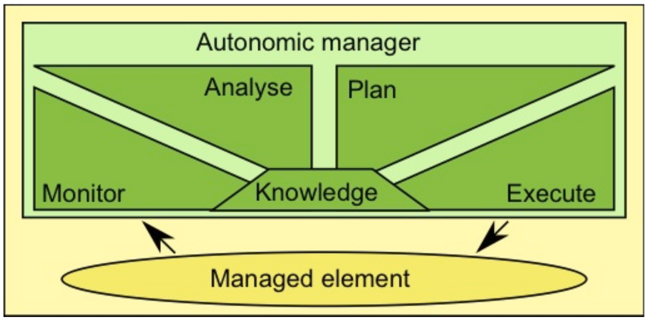

Figure 2. Autonomic systems' control loop

Our solution is an autonomous Entropy, that is capable of self-optimization (creating or deleting instances of itself so as to meet performance requirements). Thus, it has to be both aware of its environment, and adaptive to its changes. It interacts with its environment by the means of a control loop, depicted by Figure 2. This control loop regularly reports information from the environment to an autonomic manager which analyses it, takes decisions, and applies the necessary changes. This control loop is integrated with Entropy's loop which periodically retrieves the system's configuration in order to optimize it.

In our case, the information retrieved is the configuration of the system: after analysis, the control loop decides how many Entropy VM instances ,i.e., workers, are necessary to handle the retrieved configuration; execution finally creates the workers, distributes the configuration over them, and launches them.

B. Partitioning

To distribute the configuration over the virtual workers, we made the choice of having independent sub-configurations so as to avoid the very costly synchronization of the whole-system's state between our different workers. To do so, the currently implemented policy is random node partitioning, which furthermore guarantees that our workers will statistically have the same workload.

Once the number of necessary instances, that we call *groups* in the following, is computed, the nodes are randomly distributed between the different groups. This means that our partitioning is not persistent, even if two consecutive iterations require the same number of instances, the computed sub-configurations will not necessarily be the same. This is possible because Entropy does not store the system's state (mapping, CPU and memory usage), and retrieves it via the monitoring tool [3] anyway. Randomness guarantees load balancing; since the nodes are not equally populated, it guarantees that, statistically, the random groups will have about the same number of virtual machines.

Naturally, partitioning will affect the consolidation's result, and the more partitions we have, the less effective our consolidation is. It is true that load balancing significantly attenuates this effect, but we would still have an error up to the number of created partitions, on the minimum number of nodes needed to run the current virtual machines, as shown by Figure 3.

We can see that consolidating the whole cloud enables us to free two physical resources, whereas partitioning the cloud to three sub-clouds makes it impossible for this optimization to take place. However, our evaluation shows that this effect can be neglected in the context of big enough configurations, in which dynamic scalability makes sense.

C. Virtualization

The created Entropy instances are virtualized, i.e., each instance runs on a dedicated virtual machine. While it might seem that all we do is parallelize the computation as could be done using a powerful multi-core machine, we argue that virtualization for computation's sake, allows us to benefit from the cloud's resources themselves, since the created instances would be on the cloud's nodes. Besides, even pure computational tasks can benefit from virtualization as we witness the emergence of virtualized grids: Haizea [6] for instance, allows scheduling and management of virtualized jobs, i.e., jobs running on virtual machines, to benefit from the increased flexibility and reliability.

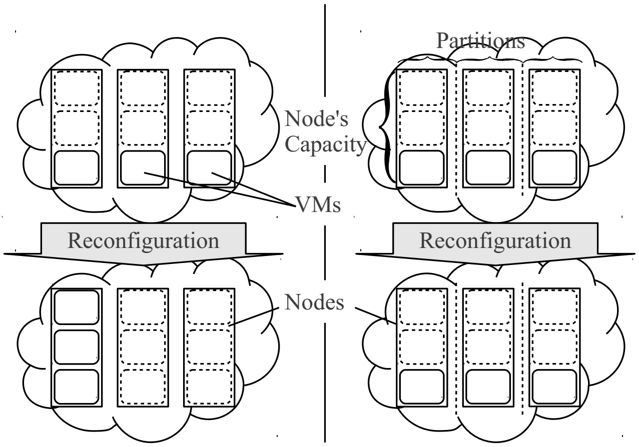

Figure 3. Partitioning's effect on consolidation

Since the cloud will contain both Entropy virtual instances and the clients' VMs and to avoid any confusion in the remaining of this paper, the term VM refers to a client's VM unless we explicitly specify that it refers to an Entropy VM instance.

D. Scaling

The estimation of necessary Entropy workers is done using the results of prior "gauging" of Entropy: we measure the performance of Entropy with gradually increasing workload and number of workers; this is done once and for all. Note that we do not rely on a predictive algorithm to pre-provision the Entropy workers, since the information that is needed to foresee the right number of instances is itself the information that is processed by Entropy. With this regard, any prediction would only shift the period of our process.

Scaling up can only be done by provisioning extra instances. However, in the case of Entropy, where two iterations are completely independent as Entropy doesn't keep knowledge of the system's state, scaling down can be done in two different ways:

- At the end of each iteration, delete all created instances. This way, every iteration will create just the right number of instances it needs.
- Instances persist after their creation, and are only deleted if they haven't been needed for the last N iterations, N to be determined empirically.

The final design is given by Figure 4. In this design, the original Entropy has been cut into two parts:

- *The Entropy Server,* that is centralized, which performs the partitioning and triggers the creation, deletion or reconfiguration commands to the hypervisors.

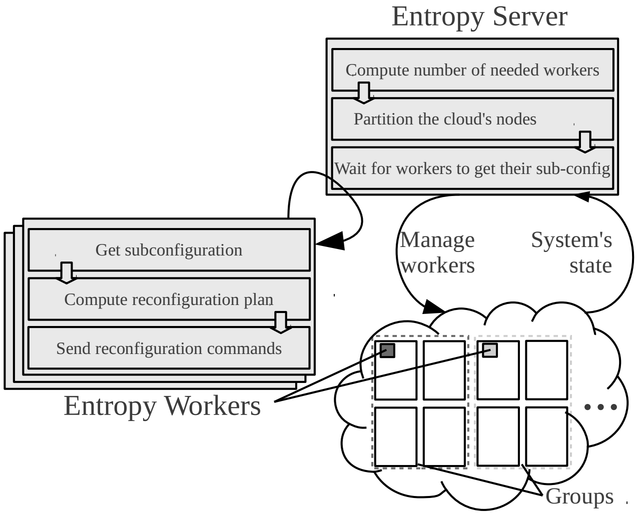
Entropy Server
Compute number of needed workers
Partition the cloud's nodes
Wait for workers to get their sub-config
Manage workers
System's state
Get subconfiguration
Compute reconfiguration plan
Send reconfiguration commands
Entropy Workers
Groups

Figure 4. Dynamically Scalable Entropy

- *The Entropy Worker,* which consolidates the sub-configuration it is assigned and issues the migration commands to its' sub-configuration's hypervisors.

The communication between the centralized part and the VM instance is done on a poll basis: once an Entropy VM instance is created, it retrieves its sub-configuration and starts processing it immediately.

E. Performance Gain

Let S be the size of the cloud (i.e., the number of virtual machines $\times$ the number of physical machines) and $Ent(S)$, the cost of the original Entropy. The cost of our virtualized version is given by:

$$Ent_{virt}(S) = Ent(S_1) + C_{virt}$$

where S_1 is the size that has been found to be the appropriate load for one entropy instance and C_{virt} is the cost of provisioning an Entropy virtual instance, it is not a function of the number of instances to be created since provisioning will not take place on the same nodes, and can thus be achieved in parallel. Moreover, this constant is amortized thanks to the relative persistence of our virtual instances.

Thus, our approach guarantees a constant cost, with only a small trade-off due to the partitioning mechanism, whereas applying Entropy to the cloud as a whole grows exponentially with its size.

IV. Evaluation

In this section, after briefly presenting the technical context and the methodology of our experiments, we discuss their results in order to validate the efficiency of our elastic consolidation manager.

A. Technical Context

The infrastructure used for our experiments is a private cloud managed by OpenStack [7], which is an EC2 compatible IaaS management solution, running on 2 racks with 6 Intel(R) Xeon(R) CPU E5645 @2.40GHz cores and 32 GB of RAM each, interconnected by a 1 Gbit/s isolated LAN. All the managed virtual machine instances are of type m1.small as described by Amazon EC2, i.e., with 2 GB of memory and 1 virtual CPU.

Note that this infrastructure is only used to host the Entropy VM instances, the configurations being consolidated are generated in order to have loads big enough to stress the scalability of Entropy.

B. Methodology

With this evaluation, our aim is to:

- see if we actually achieve any performance gain by using more workers, keeping in mind the limitation discussed in Section III-B;
- show the interest of having an elastic consolidation manager.

To do so, we consider a cloud infrastructure of 200 homogeneous physical machines, each having 3GB of memory, consuming 100W when idle and at most 200W. For this infrastructure, we generate different virtual machines which have variant requirements (CPU usage: 20%, 40% or 60%; memory: 500MB, 1GB or 2GB).

We first vary the load of our cloud infrastructure and compute the optimal consumption using 1 up to 10 Entropy workers; the load is given by:

$$load = max\left(\frac{totalMemory}{totalMemCapacity}, \frac{totalCpu}{totalCpuCapacity}\right)$$

Our metric when comparing the results given by the different sizes of workers is the error with regard to the theoretical optimal consumption, that is:

$$error = \frac{consumption - optimalConsumption}{optimalConsumption}$$

Note that the theoretical optimal consumption in which we use just enough physical machines is straightforward to have as it is given by:

$$A = totalCpu \times \frac{consMax - consMin}{100}$$

$$B = ceil\left(\frac{totalCpu}{totalCpuCapacity}\right) \times consMin$$

$$optimalConsumption = A + B$$

where A linearly computes the ratio of consumption based on the total ratio of CPU usage and B multiplies the minimum number of necessary physical machines by their minimum consumption.

What is actually hard is to find a configuration that corresponds to this optimal consumption while maintaining the constraints related to the needs of each virtual machine and the capacities of the physical machines, hence the interest of Entropy.

This first experiment will serve as a gauging on which we will base our scaling decisions for the elasticity evaluation. Our program retrieves configurations with random loads and try to compute a result within a given error margin by provisioning as many workers as needed.

Note that for all the experiments, Entropy's timeout is set to 1 minute.

C. Distributed Entropy

Figure 5 shows the results given by Entropy with different numbers of workers, for the same loads which vary from 10 to 50%.

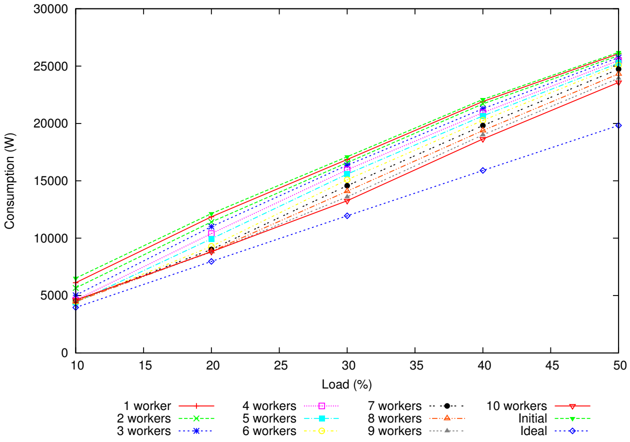

Figure 5. Comparing consumption results, for 200 PMs

We can see that for a cloud loaded at only 10%, we do not always improve the result by adding more workers which is justified by the limitation discussed in III-B. On the other hand, for more important loads, we do achieve better optimization by using more workers.

Figure 6 depicts the error rates for this experiment. Based on these values, in order to have a result within an error margin of 20%, we will need the following minimum numbers of workers, depending on the load:

Loads	10%	20%	30%	40%	50%
Workers	4	6	8	9	10

D. Elastic Entropy

Now that we have gauged Entropy, we can use it to elastically adapt to any load in order to provide a result with an error of no more than 20%. Figure 7 shows a random load profile and Figure 8 shows the corresponding behavior of our elastic consolidation manager.

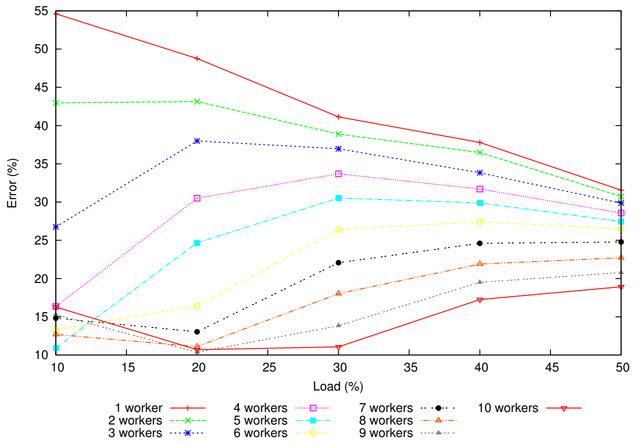

Figure 6. Comparing consumption results, for 200 PMs

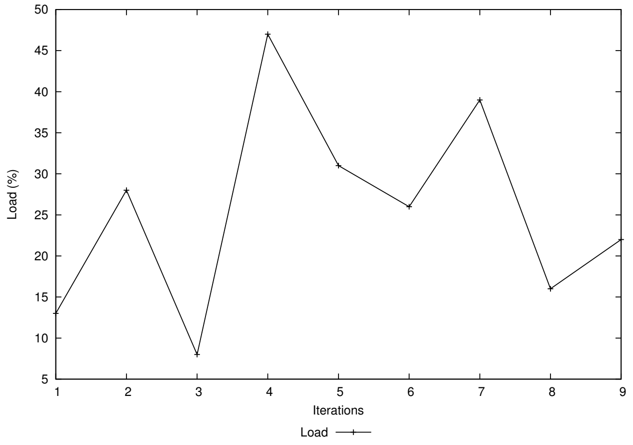

Figure 7. Random load profile

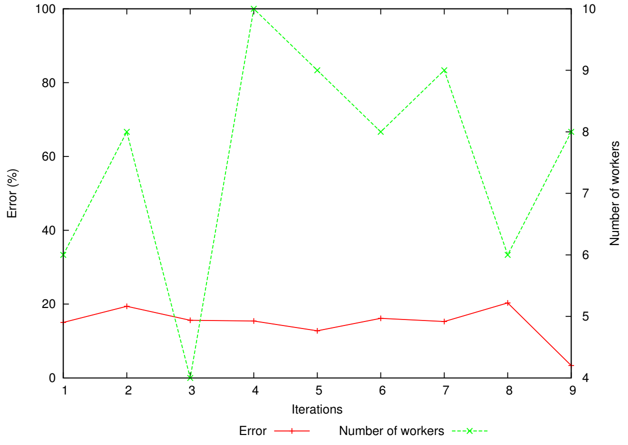

Figure 8. Elastic Entropy's error rates and number of workers

We can see that the number of workers adapts to the load in order to guarantee an error rate of less than 20%. Note that in order to have that same performance with a static provisioning, we will have to maintain 10 workers all the time although the maximum number of 10 workers is only needed once, thus the advantage of having an elastic solution.

V. Related Work

Concerning energy control, some solutions in the state of the art focus on a single node while other solutions work at the granularity of a distributed system. Most of them have initially focused on energy management for laptop powered by electric battery [8][9][10]. Other solutions [11][12] take into account cooling and thermal factors to place the workload in areas of a datacenter that are easier to cool. More recently some works addressed energy management on cluster, grid and cloud infrastructures. Most of these researches are based on (i) hardware with voltage and frequency control (e.g DVFS), (ii) resource allocation and virtualization.

In particular, many modern processors, memory chips and disk architectures embed power-saving features so that the system can adapt the energy consumption by switching hardware state through ACPI interfaces. Indeed, there is an emerging class of servers that are designed around processors with dynamic voltage scaling mechanisms (DVFS) [13]. Using this ability, a processor adjust its voltage and frequency according to the workload to optimize energy consumption independently on each node of a cluster. Researchers have developed different DVFS based algorithms and policies to save energy. Reducing power consumption by reducing the clock frequency of the processor has been widely studied [14][15]. Flautner et. al. [16] explored a software-managed dynamic voltage scaling policy that sets CPU speed on a task basis rather than by time intervals. [17] proposes a power budget guided job scheduling policy that maximizes overall job performance for a given power budget. They argue that using DVFS under a power constraint, performance can be significantly improved as it allows more jobs to run simultaneously leading to shorter wait times.

[18][19][20] focused on dynamic resource provisioning in response to dynamic workload changes. These techniques monitor workloads or other SLA metrics experienced by a server and adjust the instantaneous resources available to the server, with or without virtualization. Depending on the granularity of the server, single or replicated, the dynamically provisioned resources can be a whole virtual machine in the case of replicated servers or more fine grain resources such as CPU cycles or memory in the case of a single server. In the case of replicated servers, energy efficiency is achieved using a workload-aware, just-right dynamic provisioning mechanism and the ability to power down subsystems of a host system that are not required by the virtual machines mapped to it.

Concerning virtualization, some works address the problem at the virtual machine management level in such a way that the performance goals are met and the energy consumption is minimized. Hermenier [1][2] developed a system which relies on virtual machine migration for transparently relocating any application in the cluster. The placement policy takes into account the CPU and memory usage, in order to concentrate the workload on fewer nodes of the cluster, thus allowing unused nodes to be shutdown. Menasce[21] proposes an autonomic controller and showed how it can be used to dynamically allocate CPUs in virtualized environments with varying workload levels by optimizing a global utility function. Song[22] proposes an adaptive and dynamic scheme for adjusting resources (specifically, CPU and memory) between virtual machines on a single server to share the physical resources efficiently.

To the best of our knowledge, our work is the first to use an elastic capability at the level of the IaaS management layer itself in the case of a consolidation service. Only [23] describes a closed work in the case of migrating all virtual machines from a datacenter to another for maintenance purposes. This management operation is however driven by a human administrator. We share the basic assumption of virtualizing the management service whereas we distinguish ourselves by the real time estimation of the working set, by our probabilistic dynamic partitioning technique and by the kind of management service we addressed.

VI. Conclusion

This paper has proposed an elastic consolidation service that adjusts its size to the dynamic needs of the cloud environment. Our proposition relies on virtualizing the consolidation service, which allows easily scaling this service by provisioning dedicated virtual machines to process the consolidation. Since these virtual machines persist for a certain period of time, the provisioning cost is amortized and can thus be neglected. Any of these virtual machines is in charge of consolidating a partition of the cloud environment, defined by a probabilistic node partitioning policy. By doing so, the exponential cost of consolidation becomes quasi-constant. Whereas our solution is quite simple, a performance evaluation shows that it provides good results.

References

[1] F. Hermenier, X. Lorca, J. Menaud, G. Muller, and J. Lawall. Entropy: a consolidation manager for clusters. *Proceeding of the 2009 International Conference on Virtual Execution Environments*, 2009.

[2] F. Hermenier, J. Lawall, G. Muller. BtrPlace: A Flexible Consolidation Manager for Highly Available Applications, *IEEE Transactions on dependable and Secure Computing*, 2013.

[3] M. L. Massie, B. N. Chun, and D. E. Culler. The Ganglia distributed monitoring system: Design, implementation and experience. *Parallel Computing*, 2003.

[4] ChocoSolver home page. http://choco.emn.fr/.

[5] An architectural blueprint for autonomic computing. *IBM and Autonomic Computing*, April 2003. http://www-306.ibm.com/autonomic/pdfs/ACwpFinal.pdf.

[6] Haizea home page. http://haizea.cs.uchicago.edu/.

[7] OpenStack home page. http://www.openstack.org/.

[8] T. Halfhill. Transmeta breaks the x86 low-power barrier. *In Microprocessor Report*, February 2000.

[9] N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. S. Kim, and W. Ye. Energy-driven integrated hardware-software optimizations using simplepower. *In Proceedings of the 27th Annual International Symposium on Computer Architecture*, 2000.

[10] Alexey Rudenko, Peter Reiher, Gerald J. Popek, and Geoffrey H. Kuenning. Saving portable computer battery power through remote process execution. *In Mobile Computing and Communications Review*, 1998.

[11] J. Moore, J. Chase, P. Ranganathan, and R. Sharma. Making scheduling cool: temperature-aware workload placement in data centers. *In Proceeding of USENIX Annual Technical Conference*, 2005.

[12] C. Bash and G. Forman. Cool job allocation: Measuring the power savings of placing jobs at cooling-efficient locations in the data center. *In Proceeding of USENIX Annual Technical Conference*, 2007.

[13] A. Fox, S. Gribble, Y. Chawathe, E. Brewer, and P. Gauthier. Cluster-Based Scalable Network Services. *In Proceedings of the International Symposium on Operating Systems Principles*, October 1997.

[14] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. *In First Symposium on Operating Systems Design and Implementation*, 1994.

[15] K. Govil, E. Chan, and H. Wasserman. Comparing Algorithm for Dynamic Speed-Setting of a Low-Power CPU. *In Mobile Computing and Networking*, 1995.

[16] K. Flautner, S. Reinhardt, and T. Mudge. Automatic Performance Setting for Dynamic Voltage Scaling. *In Proceedings of the 7th ACM Int. Conf. on Mobile Computing and Networking (MOBICOM)*, July 2001.

[17] Maja Etinski, Julita Corbalan, Jesus Labarta and Mateo Valero. Optimizing Job Performance Under a Given Power Constraint In HPC Centers. *In Proceedings of the 1st International conference on Green Computing*, August 2010.

[18] Sara Bouchenak, Noel De Palma, Daniel Hagimont, Christophe Taton. Autonomic Management of Clustered Applications. *In Proceedings of the 1st International conference on cluster Computing*, 2006.

[19] J. Jaramillo, A. Quiroz, M. Parashar,F. G. Poole. Energy-Efficient Application-Aware Online Provisioning for Virtualized Clouds and Data Centers. *In Proceedings of the 1st International conference on Green Computing*, August 2010.

[20] E. Pinheiro, R. Bianchini, E.V. Carrera, and T. Heath. Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems. *In Workshop on Compilers and Operating Systems for Low Power*, September 2001.

[21] D. A. Menasce and M. N. Bennani. Autonomic virtualized environments. *In Proceeding of the Intl. Conf. on Autonomic and Autonomous Systems*, 2006,

[22] Y. Song, Y. Sun, H. Wang, and X. Song. An adaptive resource flowing scheme amongst vms in a vm-based utility computing. *In Proceeding of in Intl. Conf. on Computer and Information Technology*, 2007.

[23] Shicong Meng, Ling Liu, Ravi Soundararajan Tide: Achieving Self-Scaling in Virtualized Datacenter *In Proceeding of in Intl. Conf. on Middleware*, 2010.