An Energy Efficient Approach to Virtual Machines Management in Cloud Computing

Damien Borgetto, Patricia Stolf
IRIT, University of Toulouse, Toulouse, France
{*borgetto, stolf*}*@irit.fr*

***Abstract*—Cloud computing platforms are well established as the reference infrastructure for flexibility allowing business to scale according to user demands. However, several challenges remain to be addressed. Scaling up the infrastructure to match up the resource demand of the users is relatively straight forward. However in order to save up on power consumption, cloud providers have to dynamically consolidate the virtual machines (VM), so that only a reduced subset of hosts remains powered on. To solve this issue, the cloud scheduler, which handles the VM allocations, has to balance between providing enough resource to each user, and reducing the overall power consumption of the cloud. In this paper we investigate the VM allocation and reallocation inside a cloud in order to save energy while maintaining the resources required by users. Such actions have to be made in order to minimize the number of hosts powered on, while limiting concurrent migrations of virtual machines, and in a reasonable computational time. We propose to effectively consolidate virtual machines with an approach handling the reallocation, migration and host management problems. Our approach has been implemented in OpenNebula, and experimentally compared with its default approach.**

***Keywords*-Cloud Computing; Virtual Machine; Consolidation ; Energy-aware ; Migration**

I. Introduction

With the recent trends in application decentralization, largely embodied by cloud computing platforms, there is an increased offer and demand of applications over the internet and via mobile devices.

While traditional cloud management algorithms mainly focus on handling the initial allocation of the virtual machines, it is interesting to tackle the management of the virtual machines during their execution, as the VMs don't have necessarily the same duration. It is essential for the datacenters owners and operators to assess and improve the performance using energy efficiency. The idea being to reduce the energy consumption of the infrastructure by reducing the number of hosts powered on. One approach to dynamically reallocate resources of a cloud is to take the initial placement algorithm and periodically or upon events proceed to a full reallocation of VMs. However, it might be preferable not to have the same algorithm for initial allocation and for consolidation, since rearranging the whole pool of VMs will create a contention point where too many migrations are being done at the same time. The issue is worse if the system uses pre-copy live migration, which has less downtime but more data transfer, instead of post-copy live migration [1], which has more downtime but guarantees the memory transfer to be done only once.

That being said, the focus of the VM management approach used inside a cloud must be put on several points:

- Energy consumption: The resulting energy consumption must be lower and account for all the overheads that can impact the energy reduction.
- Migration number: The aim is to avoid a bottleneck on the network by dimensioning and limiting the migrations done in a certain time interval.
- Quality of Service: Users or applications require a certain amount of resource for their VMs, and it should be accessible to them at all time. However, even if we can respect that constraint, there are possible QoS reduction due to the time between when the VM is spawned and when it is actually running.
- Computation time: The solution must be completed in reasonable time to be enforced while it is still relevant.

The work presented in this paper has been made on the context of the SOP project (think global Services for persOnal comPuter)[1]. It aims at providing users with an all-inclusive offer that allows them to be able to use applications in the cloud. In this paper, we propose and study the different aspects of consolidation algorithms on a real platform, while comparing them to the default one proposed in OpenNebula[2], although the approach presented is not specific to OpenNebula.

We study the migration behaviour on a specific platform to better tune the consolidation algorithm. We also implement a host management strategy to be able to function with a packing based consolidation approach. Evaluation is made regarding both energy consumption and VM effective duration, while monitoring the effect of the infrastructure load on the quality of the reallocation.

This paper is organized as follows: Section II will describe related works concerning allocation and reallocation of virtual machines in a cloud computing environment. Section III and IV respectively present a problem formulation and

[1]Researches presented in this paper have been partially funded by the ANR in the context of SOP project ANR-11-INFR-001.

the approach proposed. Finally, section V will cover the experiments and discussion part.

II. Related Work

There have been extensive studies on how to manage data center and clouds in regards to the power consumption and QoS trade-off. Especially in the VM allocation and cloud reconfiguration area.

The architectural framework for energy efficient clouds proposed in [3] introduces notions of VM provisioning, as well as allocation and reallocation of VMs on physical machines. The authors use the CPU resource to dynamically consolidate the VMs, thus reducing significantly the global energy consumption of the infrastructure. Moreover, the authors implement a VM replacement policy based on a double threshold (low and high), as well as a migration minimization policy, in order to limit the number of migrations, for instance from an overloaded host. They study the impact of the Service Level Agreement degradation on energy consumption reduction.

Xiao et al. in [4] present a system based on application requirement to support green computing in the cloud. The main way to do so is by consolidation. They introduce the concept of skewness, which represent the imbalance over the different resource consumption of a server, applied in their case to the CPU, memory and disk. Reducing this imbalance is aimed at increasing individual host's utilization. The consolidation algorithm they present is based on a multi-threshold approach to define cold and hot spots. It then reallocates VMs to reduce cold spots and mitigate hot spots. The authors also combine this approach to a virtual machine load prediction to better provision the resources.

Based on the observation that the type of application running inside a virtual machine can drastically change its behaviour depending on where is the VM image, Yang et al. in [5] propose different workload characteristic-aware bin packing algorithms to be applied for consolidation in cloud environment. They propose in their paper a dynamic programming optimal algorithm and an approximation heuristic that they compare between each other, the performance metric being the number of hosts used. The approximation algorithm is a bin packing algorithm called SEP-pack, which stands for separate packing. The idea being that, since it is possible to characterize the different VMs according to their type of workload, data intensive or CPU intensive, they have to be treated differently.

In [6], the authors present two algorithms for energy consumption reduction and migration cost mitigation in the cloud. They compute an exact solution based on integer linear programming in order to minimize the energy consumption under constraints of CPU, memory and storage of the VMs. Their approach allows to assign a power budget to each host, that should not be overran. They compare it with an energy-aware best fit heuristic.

Another approach to the virtual machine packing problem by integer linear programming is also presented in [7]. The authors have worked on different VM packing algorithms that aim at reducing the energy consumption, modeled as the cost of the hosts' electrical consumption and the cost of the reconfiguration. That way, the authors aim at reducing power consumption while maintaining a certain level of performance, and diminishing the cost of reconfiguration by live migration. Reducing collisions of VM live migration is made by putting a hard constraint on the number of VMs that can migrate from and to a host. Takahashi et al. then define two algorithms, a matching-based algorithm based on graph and the modeling of the problem, that computes the optimal solution, and a greedy heuristic that computes the solution faster.

All those approaches are akin to ours, as the main focus is to save energy by powering off hosts unloaded by consolidation. However, small differences are to be seen on the sort of information the manager has, whether it is categorizing the virtual machines, being able to oversubscribe the hosts, and how to manage virtual machines migration and the bottleneck induced. The main difference being the ability to take into account the actual overhead of powering on and off hosts.

III. Problem Formulation

A. Cloud Definition

We first present the environment and the problem we are trying to solve. We are set in a cloud, operated over a single data center by a provider. This provider allows user to start and run VMs over the cloud, with the constraint of having provided to them what they have required.

The cloud has H physical hosts, with the ability to power on and off when empty. Over time, VMs are run on the cloud for a certain duration. Over the duration T of the experiment, J VMs will be executed. Each VM has resource requirements, which we chose in our case to be CPU and RAM. Let's note vm^{CPU} the number of CPU required by a VM, and vm^{MEM} the number of MB of RAM required by the VM. The VMs will be allocated on the different hosts so that their requirements are matched. We will note $e_{vm,h} = 1$ if the VM vm is allocated to the host h, $e_{vm,h} = 0$ otherwise.

If h^{CPU} is the CPU capability and h^{MEM} the MEM capability of the host, the load of the host is defined as the following:

$$r \in R = \{CPU, MEM\} : \quad load_h^r = \sum_{vm \in J} \frac{vm^r \times e_{vm,h}}{h^r}$$

The load of the host over each resource considered by the VMs should never be exceeded.

Each VM is finite in time, thus will also be defined by its duration:

$$d_{vm} = t_{sched} + t_{boot} + t_{run}$$

Where t_{sched} comprises the scheduling time and the time for its implementation. The values t_{boot} and t_{run} represent respectively the time taken by the VM boot time, and the time during which the VM is in the *running* state. It also means that if a VM requires an hour of time on the cloud, it will effectively run the requested hour minus the time taken to spawn and boot the VM.

B. Metrics

We defined a hard bound on how the different jobs are to be allocated, meaning that a VM can't get less than required. This also means, that to be able to have a sort of QoS metric, we can't reach inside the VM to get response time for example, because this model is agnostic to what kind of application the VM is running. That's why we will define the QoS metric of the VMs run on the cloud by defining the *effective duration* of the VMs:

$$d_{vm}^{eff} = \frac{t_{run}}{d_{vm}}$$

An effective duration of 90% of a particular VM will mean that this VM spent 90% of the requested time in the running state.

The other performance metric we obviously want for our model is the energy consumption of the cloud. There are two types of energy consumption that we will get or compute. The first is the real energy consumption, calculated as the sum of all the power consumptions measured over all the hosts.

$$E = \int_0^t P_t dt = \int_0^t \sum_{h \in H} P_h dt$$

The other one will be used in section V-D, is the simulated power consumption. Since the overheads of the powering on and off the hosts are quite significant as shown in [8], it can be quite hard to quantify as such the improvement of energy consumption that a consolidation algorithm will bring. As such, we can compute the simulated power consumption as the artificial elimination of all overheads induced by the host states management which is the measured power consumption with simulated hosts state. That means the assumption that the hosts power on and off instantly, and with no cost.

$$E^{sim} = \int_0^t P_t^{sim} dt = \int_0^t \sum_{h \in H} \sum_{vm \in J} e_{vm,h} \times P_t dt$$

IV. VM Management Approach

A. OpenNebula Scheduler

OpenNebula already provides a base scheduler with its installation. It is a configurable scheduler called *match-making scheduler* or *mm_sched* and is described in details in [9].

The *mm_sched* algorithm operates in three simple steps:

- Filter hosts that lack the required characteristics (e.g: that are not in the right state or don't have enough resources).
- Sort all hosts according to the ranking policy (described hereafter).
- The hosts with the highest rank are chosen for the allocation.

B. SOPVP

The implementation of the scheduler for the cloud that we defined will follow the same functioning pattern as *mm_sched*. We will have the same periodic calls to the scheduler in order to have a consistent comparison.

The scheduler has to take into account different constraints. The first is that it has to be separated into two distinct algorithms: one for the initial allocation of VMs, and one for the periodic reconfiguration of the VMs in the cloud, to better handle the subset of migrations that have to be done for energy reduction without having to reallocate all VMs. The algorithm allocates pending VMs based on a best-fit heuristic.

The algorithm to reallocate will bring the most energy savings by using consolidation to reduce the number of hosts used by the VMs when the system changed because some of the VMs have ended for instance. We used a modified vector packing algorithm, described as algorithm 1, to reallocate the virtual machines from one or several hosts to reduce the global host number. The approach used is based on the algorithm presented in [10], and is working by separating the hosts in as much list as we have dimensions for the allocation. Each list contains the hosts with more requirements in one particular resource. In our case, we will have two lists, one for the hosts that have more CPU than MEM, and one for the opposite. The CPU and MEM are not the same type of resource, since the CPU can be oversubscribed more easily than the memory, and it is better to not provide an application with less memory than what it required. However, we took into account both resources at the same level to guarantee an execution time quality of service for applications running inside VMs.

Then, the VMs are allocated to the hosts, previously sorted, so that the allocation reduces the current resource imbalance. For example, if the VM has $vm^{CPU} > vm^{MEM}$, then we will pick preferably a host from the list where $load_h^{MEM} > load_h^{CPU}$. That way, the imbalance of resource consumption will be reduced.

We have to choose the number of hosts that we will try to consolidate. That number can be chosen via numerous ways, like having thresholds to define low loaded hosts that will be chosen to consolidate, or like we do in this paper choosing to unload a fixed number of hosts. That number must vary regarding the global number of hosts and VMs. In addition, it will also have to depend on the number of concurrent migrations that can be made toward one host, and on the network infrastructure. Live migrations are putting a strain on the network as they transfer the memory pages of the VM potentially more than once, depending on how data intensive is the application. That being said, the number of hosts chosen for unloading needs to be capped to the maximum amount of concurrent migration one wants, as well as being tailored to match the infrastructure size.

We will compute the number of hosts to be 20% of the total host number H, with a minimum of 1. We also chose to mitigate that number regarding the projected amount of migrations, as we will describe in V-C. The algorithm 1 is presented for a number of 1, since it's rather straightforward to extend it to several hosts to unload.

As we can see, in the algorithm, one should note that the migrations are only enforced if and only if the host may be fully unloaded. It is an optimization choice. The aim is to avoid migrations that could irrelevant because another decisions may be better in the next scheduling loop and that will only generate overhead.

C. Implementing Energy Savings

There are several ways to handle the hosts state, that is when to power off and power on the physical machines to better save energy accordingly to what the VM consolidation algorithm is doing [8].

1) FirstEmpty: The first one is the easiest but not the most efficient, it works in a simple way, switching off hosts as soon as one host is empty, and power on hosts when the global load of the cloud is over a certain threshold.

2) Pivot: The second one works by keeping a pivot host. It first computes the projected number of host that is needed to run all the VMs, and tries to keep this number of hosts powered on plus the number of pivot hosts. This reveals the fact that the algorithm used for consolidation has to account for the pivot host, otherwise it will wind up consolidating too much and having an empty host, as previously shown in [11]. Although in any case it allows some leeway in the reallocation process, which will diminish VM effective duration degradation.

$$host\_number = \max_{R}\left(\frac{\sum_{vm \in J} vm^{r}}{H}\right)$$

3) Variation: The last host state management strategy we defined is based on the variation of the global load over time. It keeps the history of all the variations that occurred, and count how many of the loads are going up and down. If there is more down than up, it will try to power off a host, and vice versa.

begin Consolidate VMs
  H = sort(H, load ascending);
  llh = $h \in H$ where min($load_h^r$);
  $\{H^{CPU}, H^{MEM}, Migrations\} = \emptyset$;
  for $h \in H, h \neq llh$ **do**
    if $load_h^{CPU} > load_h^{MEM}$ **then**
      add h to H^{CPU};
    else
      add h to H^{MEM};
    end
  end
  success=True;
  foreach $vm : (e_{vm,llh} = 1)$ **do**
    found = False ;
    if $vm^{CPU} > vm^{MEM}$ **then**
      found = Try to find a suitable host h in H^{MEM} first, then H^{CPU};
    else
      found = Try to find a suitable host in H^{CPU} first, then H^{MEM};
    end
    if $found$ **then**
      add (vm, h) to $Migrations$;
      change list of h if needed ;
    else
      success=False;
    end
  end
  if *success=True* **then**
    enforce $Migrations$;
  end
end

Algorithm 1: SOPVP Reallocation algorithm

V. Experiments

A. Experimental Platform

In this section, we will describe the experimental platform and methodology used to evaluate the allocation and reallocation strategies, and compare them against each other. We have done the experiments on a RECS testbed [12]. It comprises a single cloud deployed with OpenNebula over KVM, on 6 Intel i7-3615QE nodes (4 cores). Power consumption values are retrieved for each single processor with a Plogg watt-meter, thus giving us the ability to monitor each host (i.e: an i7) separately. In regards to OpenNebula, VM images are stored in a shared NFS storage, and each host has a capacity of 4 CPU, and 15.6 GB of RAM.

B. Methodology

Each experiment is done over the different scheduling algorithms. The first is the one that is default in OpenNebula, *mm_sched* with the striping heuristic. This setting is set by default because it is the one that will give the best QoS to the VMs, as it tries to allocate each VM to the host where there is the most resources.

The SOPVP scheduler is described in IV-B. For each algorithm, we collect several metrics. The power consumption as expected, but also the migrations number over time, and finally the relative duration of the VMs. The relative duration is computed as the actual duration from the moment the VM is actually running on the cloud, divided by the total duration, which is the duration from the spawn of the VM, until its deletion.

We randomly generate the workload over time on the cloud by randomly spawning VMs that are CPU stressed for their running duration. The random generation is done following a Poisson process, with a λ value tailored to generate an average load onto the cloud. The mean load of the cloud depends on the mean load generated by each VM. We will generate each VM to take between 0.1 and 2 CPUs, following a uniform distribution. Each VM will last for a duration between 300 and 500 seconds, uniformly generated, which is an average duration of 400 seconds. We will also generate the memory requirement for each VM to be between 1024 MB and 6624 MB. Each VM is designed to represent on average around 25% of a single host.

With those values in mind, we can compute the λ required for the Poisson process that will generate a specific average load.

Each VM is submitted to the cloud for the duration of the experiment, and the schedulers, that are on a periodic loop of 30 seconds, will allocate and reallocate them. Experiments between different schedulers are done using the same generated workload. Each set of settings is used on an experiment that will last 2 hours. That means that we've done a total of 17 different experiments, which represents more than 34 hours.

One should note here that a lot of variability inside a single run can occur. Indeed, there are small variations in the orders of actions that can take place when for example OpenNebula is responding to request, or even the variability in the time to boot a VM or a host, and yield to a different result.

C. Concurrent Migrations

We first start with an empirical experiment to calibrate the number of migrations we can do simultaneously on the testbed. To do so, we instantiate several VMs each with 2GB of memory, then we choose a fixed number of VM and migrate toward a random host. Obviously the network layout and the application inside the VM plays a big part in the result, but we need to keep in mind that it is only to define the number of hosts that we will unload in the algorithm. The RECS's hosts are all connected to a 1 Gbps switch. The last part of the data transfer, when the VM is stopped on the source host and the core of dirtied page is sent to the destination host [13], is done without bandwidth limitation. Theoretically, if all the memory of the VM is dirtied during the process of sending memory pages, migrating a single VM would take up to 18 seconds.

As we can see in Table I, the average migration time of the virtual machines migrating concurrently from one host to another is increasing as the number of migration increases. This is possibly due to either the speed of the memory of the destination host, some contention in the network, and the time OpenNebula takes to proceed with the migrations.

Although the numbers themselves are specific to the platform and experimentation, it still tells us that at a certain point, concurrent migrations tend to be near a time that is unacceptable. This time is defined arbitrarily by the provider.

In our platform with this particular setup and VM size, we consider that 6 concurrent VM live migrations toward the same host is the maximum we aim to set.

Table I
TIME SPEND WITH CONCURRENT LIVE MIGRATIONS

Nb. migrations	1	2	3	4	5	6
Avg. Time (s)	5.5	7.44	11.68	12.55	14.05	16.62

D. Simulated Energy Consumption

In this experiment we have run the workload of VMs on the cloud without powering on and off hosts, but we have used the simulated power P_t^{sim} described in section III-B. As we can see in Figure 1 which plots the energy consumption on different cloud loads. We can see that there is a steady increase of the energy consumption due to the increase in load, and that SOPVP is more energy efficient by between 36% and 16% for respectively low and high loads.

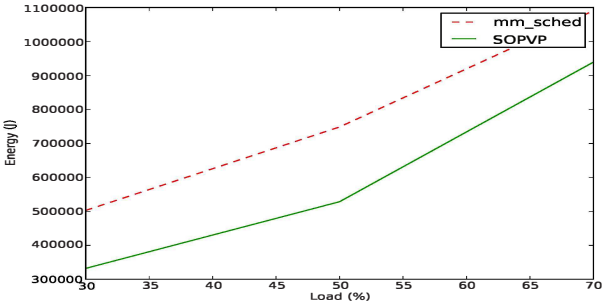

Figure 1. Power consumption vs System Load, simulated energy consumption

However, those values are mitigated with the results shown in Figure 2 which plots the effective duration of

the VMs. As expected, with an increase in load we see a decrease in average VM duration. This is due to the spikes in load that render the initial allocation harder to achieve. We can also see that even if the SOPVP is better at low loads, the difference between the two decreases as the load increases, to be even in reverse roles at high loads, because mainly SOPVP does more requests to OpenNebula, thus slowing a bit the decision by waiting each time for the responses.

We can see however that the duration is between 0% and 3% of average VM duration, which is rather low, thus meaning energy savings can be achieved without QoS degradation if host management overhead is neglected.

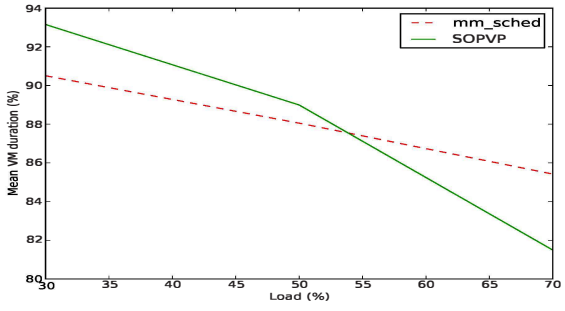

Figure 2. VM durations vs System Load, simulated energy consumption

E. Host Management Strategy

In this experiment, we are trying out the different types of host management strategies as described in IV-C, and their effect on the energy consumption and average effective duration. Table II presents the results of both the energy consumption, where lower is better, and the effective duration, where a higher value is better. NOHM represents the baseline when we are not switching on and off the hosts. The PIVOTCEIL is similar to the PIVOT strategy with the number ceiled instead of rounded. The values are for a cloud with an average load of 50%, and the same SOPVP algorithm, which as seen before consolidates effectively. As we can see in Table II, the most efficient strategy is the pivot, as it is the most tailored for this experiment in particular, and reacts the best to fast and slow load changes. We can save up to 23% of power while having a degradation of QoS of only 5%. We can also see that compared to the FIRSTEMPTY approach, we gain with PIVOT both in energy consumption (7%) and QoS (4%). The VARIATION strategy yields bad results, since for this kind of cloud load, it will switch on and off hosts, when there are already a sufficient number of host usable.

F. Real Energy Consumption

In the last experiments we have implemented real powering on and off of the hosts, to see whether the energy consumption reduction observed in the simulated power consumption experiments remains. This time, the overhead in both time and power consumption due to the powering on and off of the hosts is not ignored, and it can be significant, as shown in [8]. We will conduct experiments twice as long as in V-D, with the host management strategy PIVOT presented before.

Table II
HOST MANAGEMENT STRATEGIES

Strategy	noHM	FirstEmpty	Pivot	PivotCeil	Variation
Energy (kJ)	467	360	334	398	481
Duration (%)	84.58	77.24	80.73	88.86	87.40

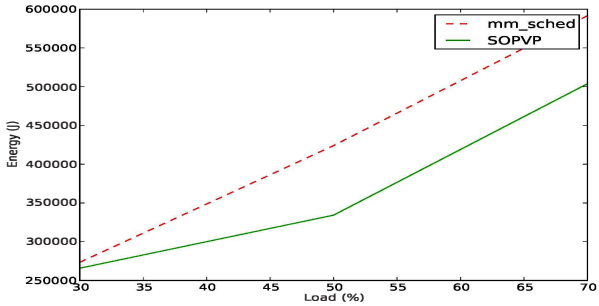

Figure 3. Energy consumption vs System Load, real energy consumption

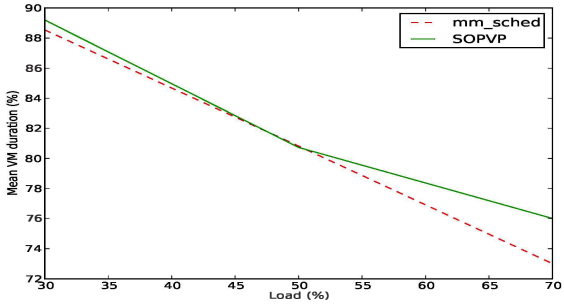

Figure 4. VM durations vs System Load, real energy consumption

As we can see in Figure 3, which represents the energy consumption for loads of 30%, 50% and 70%, the energy consumed is still better using the SOPVP algorithm. However, for loads of 30%, the energy consumption difference between the two algorithms is less than what was achieved using the simulated power consumption, which is because when the mean load is around this value, it oscillates between 10% and 50%, which means that since we have only 6 hosts, it is difficult to go with a load under than 2 hosts. It also means that in proportion the overhead for switching on and off an host will have more impact.

We can also see that for higher loads, we can still save up 21% energy for a load of 50%, due to the reasons mentioned above, but also to a higher strain on the cloud, it is even more important to power on and off hosts.

Figure 4 presents the corresponding average effective duration of the virtual machines. We can see that we can maintain roughly the same amount of QoS as mm_sched, but that QoS is dropping as the load increases, which is to be expected. This drop is due to different reasons. For mm_sched, it's mainly due to the fact that on one hand we have the PIVOT, which may be suited because it will take advantage always of the extra host if any, but on the other hand the global load on the powered on hosts which is higher induces the algorithm to fail to find suitable hosts for VMs, which leads to pending VMs, waiting to be allocated. That's also the reason why SOPVP is better at high loads, because there are more VMs, and it becomes increasing more likely that mm_sched will not find suitable host.

One should note here that even if an effective duration around 70-80% in average seems bad, it is mainly due to the fact that with VM duration around 400 seconds, a scheduling loop of 30s, 30s to power on an host, and between 30s and 1 minute for OpenNebula to acknowledge the host and add in to the host pool we can have a important impact of the waiting time that is due to the experimental platform, since it represents a high portion compared to the average duration. Such an effect would be greatly mitigated it we took VMs with higher mean durations.

VI. Conclusion and Future Works

In this paper we have studied the reallocation of VMs in a cloud, in order to save energy. We have compared the algorithm we proposed, SOPVP to the default algorithm provided in OpenNebula. We took into account several important factors to achieve consistent consolidation, such as the number of hosts to unload, directly related to the number of migrations we aim to make, but also the overhead induced by the powering on and off of the hosts, which takes time by itself in a real system.

We aim at pushing the evaluation further by tackling the computation time, which is low with the SOPVP, but that we could increase by adding steps in larger instances to increase the quality of the reconfiguration, while keeping a low computation time. However, this must be done on much larger instances, thus needing simulation. For this purpose, we aim to use the DcWORMS[14] simulation platform to model and simulate the execution at a larger scale. The host management strategies are also critical and must scale and behave properly with larger instances, and will need refining. Finally, we have to switch from artificial workloads to real workloads, to better match real world environments.

References

[1] M. R. Hines and K. Gopalan, "Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning," in *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, ser. VEE '09. New York, NY, USA: ACM, 2009, pp. 51–60.

[2] "Open nebula: Flexible enterprise cloud made simple," http://opennebula.org/, June 2014.

[3] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Gener. Comput. Syst.*, vol. 28, no. 5, pp. 755–768, May 2012.

[4] Z. Xiao, W. Song, and Q. Chen, "Dynamic resource allocation using virtual machines for cloud computing environment," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 24, no. 6, pp. 1107–1117, June 2013.

[5] J.-S. Yang, P. Liu, and J.-J. Wu, "Workload characteristics-aware virtual machine consolidation algorithms," in *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, Dec 2012, pp. 42–49.

[6] C. Ghribi, M. Hadji, and D. Zeghlache, "Energy efficient vm scheduling for cloud data centers: Exact allocation and migration algorithms," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, May 2013, pp. 671–678.

[7] S. Takahashi, A. Takefusa, M. Shigeno, H. Nakada, T. Kudoh, and A. Yoshise, "Virtual machine packing algorithms for lower power consumption," in *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, Dec 2012, pp. 161–168.

[8] L. Lefèvre and A.-C. Orgerie, "Designing and Evaluating an Energy Efficient Cloud," *Journal of Supercomputing*, vol. 51, no. 3, pp. 352–373, Mar. 2010.

[9] "Open nebula: Match making scheduler," http://archives.opennebula.org/documentation:rel4.4:schg, June 2014.

[10] W. Leinberger, G. Karypis, and V. Kumar, "Multi-capacity bin packing algorithms with applications to job scheduling under multiple constraints," in *Parallel Processing, 1999. Proceedings. 1999 International Conference on*, 1999, pp. 404–412.

[11] D. Borgetto, M. Maurer, G. Da Costa, I. Brandic, and J.-M. Pierson, "Energy-efficient and SLA-Aware Management of IaaS Clouds (regular paper)," in *ACM/IEEE International Conference on Energy-Efficient Computing and Networking (e-Energy), Madrid, 09/05/2012-11/05/2012.* ACM DL, 2012.

[12] "The RECS testbed," http://recs.irit.fr/doku.php?id=recs:features, June 2014.

[13] "Kvm live migration," http://www.linux-kvm.org/page/Migration, June 2014.

[14] K. Kurowski, A. Oleksiak, W. Piatek, T. Piontek, A. W. Przybyszewski, and J. Weglarz, "Dcworms - a tool for simulation of energy efficiency in distributed computing infrastructures." *Simulation Modelling Practice and Theory*, vol. 39, pp. 135–151, 2013.