Host management policy for energy efficient dynamic allocation

Patricia Stolf, Damien Borgetto, Mael Aubert
IRIT, University of Toulouse, France
{patricia.stolf, damien.borgetto, mael.aubert}@irit.fr

***Abstract*—Nowadays, reducing the energy consumption of large scale and distributed infrastructures has truly become a challenge for both industry and academia. Dynamic job allocation and resources provisioning is important to fit users requirements. The aim is to minimize the number of hosts utilized in order to reduce the energy consumption while maintaining a correct level of quality of service for the users. Green leverages like migration and on/off actions have cost overheads. Switching on and off a host has a cost: it takes time and it consumes power. These actions have to be optimized and should anticipate load variations. In this paper we investigate host management linked with the allocation and reallocation algorithm in order to optimize the number of powered on hosts and to reduce the overheads. We propose an original host management algorithm based on a genetic algorithm. Our approach has been implemented in DCWoRMS simulator and compared with other heuristics.**

***Keywords*-Scheduling, energy consumption, host management, green leverages**

I. Introduction

IT systems and infrastructures have one of the fastest growing energy consumption. Since 2007, the Green500 [1] ranks the supercomputers based on the flops per watt metric. The supercomputer at the top of the list in June 2016 has a performance-to-power ratio of 6.6738 GFLOPS per Watt. In 2011 Koomey [2] reports that total electricity use by datacenters in 2010 of about 1.3% of all electricity use for the world, and 2% of all electricity use for the US. Heddeghem [3] estimates that worldwide datacenters have consumed up to 270 terawatt-hours (TWh) in 2012, which accounts for almost 2% of global energy consumption. Greenpeace [4] reports energy consumption in datacenters and analyzes cloud energy consumption. If compared with the electricity demand of countries in 2011, the cloud would rank 6th in the world, with demand expected to increase 63% by 2020. Energy efficiency is a challenge to reduce the consumption in datacenters. The energy efficiency is studied in the literature at various levels [5] [6]: at the node level or at the infrastructure level. Approaches can be static or dynamic (power scaling processors, power scaling memory, load balancing ...). In this paper we present a dynamic and pro-active approach. We consider a HPC workload taking benefits from the virtualization and virtual machine migration provided by cloud technologies. Different articles [7], [8] have studied the cloud-based environments to HPC architectures: HPC2 [9]. HPC is historically known to suffer from performance degradation in cloud deployments. However, improvements in virtualization technologies have significantly reduced the performance gap between physical and virtual deployments. It is therefore not surprising that HPC users are shifting some workloads to cloud in order to benefit from flexibility, cost efficiencies and improved resource sharing that cloud provides. For example, the Xlcloud [10] project aims to provide tools that facilitates HPC in cloud deployments. We propose to improve an existing greedy heuristic for virtual machines allocation with a new host management policy. The aim is to limit energy and time overheads of machines switches on and off. In [11] we have presented a consolidation heuristic handling the reallocation, the migration and host management issues. Work in [11] has been extented in two ways: first the heuristic has been evaluated in large scale with simulations ; second, a new host management policy has been proposed based on a genetic algorithm.

The rest of this paper is organized as follows. Section II formally states the problem we address. Section III presents the host management heuristic we propose, followed by the simulation results in section IV. Section V presents the related work and Section VI concludes the paper with some future directions.

II. Problem Statement

We consider a datacenter with H physical hosts and J running VMs (which we will also call tasks) comprising HPC jobs with known required execution time. Each VM has resource requirements, which we choose 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. Let's note h^{CPU} the CPU capability and h^{MEM} the MEM capability of a host h. VMs are allocated and periodically reallocated with a greedy heuristic. In [11] we proposed *SOPVP* approach which is an initial allocation with a bestfit algorithm and a periodic reallocation algorithm based on a vector packing algorithm to reallocate VMs from one or several hosts in order to consolidate the load while minimizing the network contention at fixed time intervals. The algorithm uses a host management algorithm called *Pivot* [11] to handle the hosts states (i.e. to decide when switching on and off the hosts) to enforce energy savings computed by the reallocation heuristic. The *Pivot* heuristic first computes the theoretical number of hosts that are needed to run all the VMs, and tries to keep this number of hosts powered on plus a small amount of over-provisioning. The theoretical host number is computed as the sum of maximum requested resources per VM divided by the average resources available per host with the following

equation:

$$host\_number = max(\frac{\sum_{vm \in J} vm^{CPU}}{\frac{\sum_{h \in H} h^{CPU}}{H}}, \frac{\sum_{vm \in J} vm^{MEM}}{\frac{\sum_{h \in H} h^{MEM}}{H}}) \quad (1)$$

For the homogeneous case we study here, this equation becomes:

$$host\_number = max(\frac{\sum_{vm \in J} vm^{CPU}}{h^{CPU}}, \frac{\sum_{vm \in J} vm^{MEM}}{h^{MEM}}) \quad (2)$$

In this article, we try to improve the estimation of number of hosts to power on. Our aim is to find a formula based on the past to predict how many hosts must be powered on as depicted in figure 1. This formula will be used by the mapping algorithm to optimize the virtual machines placement. To do that, we use a fixed time window. The first half of the window is used to find the formula based on past collected data: some monitoring information about the machines states and about the virtual machines. Then a genetic algorithm is used to find the best formula. The formula is then applied during the time steps of the second half of the window with the new state of the datacenter and virtual machines. The formula is built on a set of variables describing the system and a set of operators. The aim of the genetic algorithm is to find the best formula that can be applied. The formula will be recomputed at each time window ending.

III. Genetic Algorithm for host management

A genetic algorithm (GA) mimics the process of natural selection [12]. This metaheuristic is used to generate solutions to optimization and search problems. It belongs to the larger class of evolutionary algorithms, which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover.

In this section we will describe the genetic representation, the operations, the fitness function and the algorithm.

A. Genetic representation

The individuals in our approach represent the formula to compute how many hosts to switch on in a datacenter infrastructure. We will note each individual $f()$. Each individual is a formula composed of two chromosomes.

a) Chromosome 1: Characteristics of the system: The first chromosome X is a complete set of variables describing the system as the number of machines, the number of VMs (or tasks) or the CPU load of a host h at time t. This set of variables is described in table I. Each variable will have a ponderation coefficient called $\alpha \in [0, H]$ and each variable will have a value for each time step of the time window considered. t represents the current state when applying the formula $X = \alpha_1 X_1, ..., \alpha_n X_n$. Each variable can have a timestamp between t and $t - w$ where w is the time window length.

There are two types of variables that we want to exclude. First, genes whose values are zero in the historical data will not be included in an individual. Second, genes relatives to virtual machines whose time to completion is close will not be included in an individual (in our implementation we have chosen not to include VMs which have a TTL greater than 85%). This will avoid a formula which could be relevant at time t but which will overestimate or underestimate the number of hosts to power on at time $t + 1$. For example, we could have the following chromosome X described in table II where H represents the number of hosts in the cloud, $load^{CPU}_{h1,t-1}$ is the CPU load of host $h1$ at the last timestamp, J_t is the number of tasks and $load^{MEM}_{h2,t-2}$ is the memory load of host $h2$ at time $t - 2$.

TABLE I: Variables for chromosome X.

H	Number of hosts
H^{ON}_t	Number of hosts powered on at time t
J_t	Number of tasks at time t
$load\_cloud^{CPU}_t$	CPU load of the cloud at time t
$load\_cloud^{MEM}_t$	Memory load of the cloud at time t
$load^{CPU}_{h,t}$	CPU load of host h at time t
$load^{MEM}_{h,t}$	Memory load of host h at time t
$TTL_{j,t}$	Percentage of completion of VM j at time t
vm^{cpureq}_j	CPU request of VM j
vm^{memreq}_j	Memory request of VM j

b) Chromosome 2: operators: The second chromosome Y has $n - 1$ genes, each representing an operator for the formula $f()$. Each operator o can be $+$, $-$ or $\times$.

TABLE II: Example of chromosomes X and Y.

X	$0.5H$	$12load^{CPU}_{h1,t-1}$	$5J_t$	$2load^{MEM}_{h2,t-2}$
Y	$+$	$-$	$\times$	

c) The individual genes: An individual $f(X, Y)$ is built on the two chromosomes X and Y:

$$f(X, Y) = \alpha_1 X_1 o_1 \alpha_2 X_2 ... o_{n-1} \alpha_n X_n$$

Based on the examples given, we would have the following individual:

$$f_{example}(X, Y) = 0.5H + 12load^{CPU}_{h1,t-1} - 5J_t \times 2load^{MEM}_{h2,t-2}$$

B. Genetic algorithm operations

During individuals' reproduction, the chromosomes of both parents merge. This fusion results in the creation of a genotype child constituted by the genotype of both parents. Evolutionary principles allow for three operations to take place amongst individuals, and allow for their diversity. These operations are imitated by the genetic algorithms to maintain the diversity of the populations and thus be able to find better solutions and make converge the solutions on an optimal solution. So in a genetic algorithm, a population of candidate solutions is evolved toward better solutions. Each candidate solution can be mutated and altered through genetic operators: mutations, crossover and selection.

1) Mutation: In each generation a gene has a probability to be modified by the mutation operator. In our genetic representation, each chromosome could be altered. For chromosome

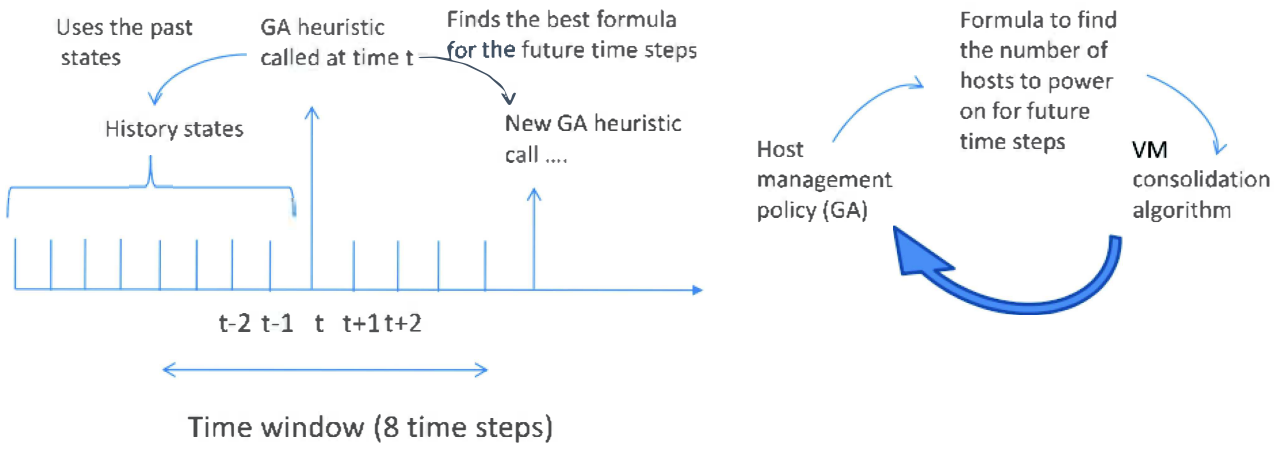


Fig. 1: Host management principle

X, the couple $\alpha_i X_i$ could be affected and for chromosome Y, an operator could be modified.

$$mutation_X = \alpha_i X_i \longrightarrow \alpha_j X_j$$
$$mutation_Y = o_a \longrightarrow o_b$$

2) Crossover: During the crossover operation, the chromosomes of both parents exchange a part of their genes to generate two children. A point is thus chosen randomly to be able to separate chromosomes into two chains. In our genetic representation, the crossover point is randomly chosen on the entire individual constructed with the concatenation of chromosome X and Y.

3) Selection: The selection process aims at keeping only individuals inclined to provide better results. For the genetic algorithms, it will be a question of sorting out the solutions in order to keep only the best solutions. The selection will be made by tournament, which is a common algorithm used in the literature [13]. Every individual is assigned a fitness value which is the capacity of an individual to be close to the ideal solution. The fitness function is computed on a given history window. The fitness represents the distance of the values expressed by an individual to the theoretical ideal values. These theoretical values are given by the execution of a placement algorithm in an ideal infrastructure with an infinite number of hosts at time $t+1$ and without any cost overheads for machines switches on/off. We call Z the heuristic chosen to estimate the theoretical number of hosts (based on a valuable history of the real system) and w the size of the time window. The fitness function computes the accumulated absolute error, the goal is to minimize its value. It is computed with the following equation:

$$fitness = \sum_{i=t-(\frac{w}{2})}^{t-1} |f(X_{t-i}, Y_{t-i}) - Z_{t-i+1}|$$

```
population = initialize_population();
for i ∈ GENERATION_NUMBER do
    new_population = [];
    new_population[0] =
     get_fittest_individual(population);
    mating_pool = selection_tournament(population,
     T_SIZE);
    for j ∈ POPULATION_SIZE do
        parent1, parent2 = mating_pool[random_int];
        if CROSSOVER_PROBABILITY then
            child1, child2 = crossover(parent1, parent2);
            new_population[j] = choose one child at
             random;
        end
        new_population[j] = parent1;
    end
    for individual ∈ new_population do
        if GA_MUTATE_X_PROBABILITY then
            individual.mutateX();
        end
        if GA_MUTATE_Y_PROBABILITY then
            individual.mutateY();
        end
        individual.fitness(x, history);
    end
    population = new_population;
end
```

Algorithm 1: Host management algorithm

C. Algorithm

The initial population is randomly generated. Then, the algorithm is an iterative process. The population in each iteration is called a generation. In each generation, the fitness of every individual in the population is evaluated on the first half of the window. The individuals are evaluated and ranked. Then crossover and mutation are randomly performed to alter existing individuals and generate offspring genes. The new

generation of candidate solutions is then used in the next iteration of the algorithm. The algorithm terminates when a maximum number of generations has been produced. The fittest individual will be the formula that we will apply on the cloud data to obtain a prediction of the number of hosts for the following timesteps. The algorithm is described in algorithm 1 and uses the variables described in table III.

IV. Evaluation

To evaluate the approach we have used the DCWoRMS simulator [14]. Each job is defined by a request in CPU, MEM and an arrival time. As a QoS metric we choose the waiting time which is the time between a task arrival in the system, and the beginning of its execution. The second metric we choose is the power consumption of a node defined as:

$$P_h = (P_h^{max} - P_h^{min}) \times load_h + P_h^{min}$$

where P_h^{max} is the maximum power consumption of the host h, and P_h^{min} is its idle consumption. We use a randomly generated workload over time. We simulate a datacenter of 100 nodes, each with 8 processors and 16 GB of RAM. P_h^{min} and P_h^{max} have been chosen to be respectively 70W and 140W, values hand picked on the SPEC Power web page [15]. The workload is comprised of tasks lasting from 500 seconds to 3500 seconds. They require between 0.25 and 3.75 processors, as well as between 1024 and 7168 MB of RAM. All those values are randomly chosen following a uniform distribution. The inter arrival of tasks follows a Poisson process, chosen to load at the peak of the experiment to a certain percentage. The experiment starts with the first task arrival and ends with the departure of the last one. At the beginning all hosts are powered on. We have implemented in DCWoRMS the *BestFit* approach, with no reallocation, and the *SOPVP* approach [11] with the *Pivot* host management algorithm and the *SOPVPGA*. We call *SOPVPGA* the resulting algorithm coupling the *SOPVP* heuristic for the reallocation of VMs and the genetic algorithm for the host management policy. The *BestFit* allocation is described in algorithm 2.

```
begin Allocate Pending VMs
    Data: pendingVMs
    Data: H
    Sort VMs, highest MEM first;
    foreach vm ∈ pendingVMs do
        h' = ∅;
        foreach h ∈ H do
            if vmCanFitOnHost(vm, h) then
                if h' = ∅ then
                    h' = h;
                else
                    if max(load^r_h) ≥ max(load^r_h') then
                        h' = h;
                    end
                end
            end
        end
        if h' ≠ ∅ then
            allocate(vm, h');
        end
    end
end
```

Algorithm 2: *BestFit* Algorithm for VMs allocation

A. Host management strategy evaluation

The genetic algorithm for the host management policy uses a time window. We have studied the impact of this parameter. For an infrastructure load of 50% at each scheduler iteration we compared the theoretical number of hosts that is needed to run all the VMs with the number of hosts given by the GA host management heuristic for four time window lengths: 4, 8, 16, 24. The population has 150 individuals, for the tournament selection there are 75 individuals, the crossover probability is 50%, the mutation probability for each chromosome is 10% and there are 60 generations. In figure 2 we see that the wider the GA window is, the less precise the predictions are and less they follow the desired results. A good compromise is obtained with a time window equal to 8 since we observe good precisions without too many calls to the GA heuristic. For all the following results we have chosen this window size.

There is a strong coupling between the host management strategy and the scheduling algorithm when the goal is to improve energy efficiency. On one hand powering off and on hosts will make them unavailable for a time, thus potentially preventing tasks to run. On the other hand, if the reallocation algorithm consolidates tasks on a reduced subset of hosts, and the hosts are not powered off by the host management strategy, the reallocation would just be a waste of resources.

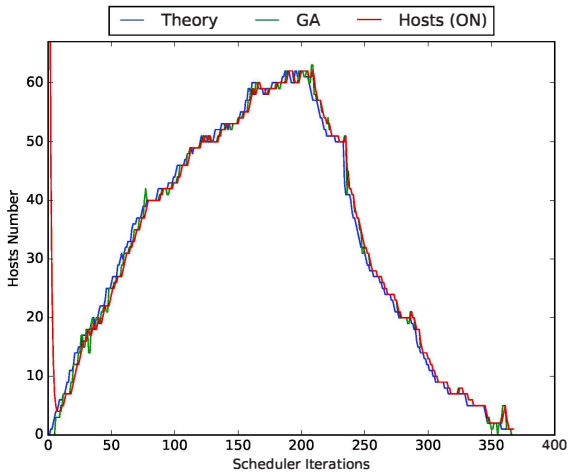
B. Scheduling Simulations

We have evaluated the energy consumption and the waiting time for each load between 10% and 80%, with a step of 10%. Each experimentation has been reproduced 10 times for each load and each algorithm. Live migration is a complex process, that highly depends on numerous factors like the type of VM, its size, memory dirtying rate, as well as the network link size on both source and destination host. That's why it is usually poorly modelled inside simulators. With DCWoRMS, even though we are able to migrate tasks between nodes, we are only able to model the cost of the said migration through task delay. This means that instead of having a live migration of tasks, we will only have cold migration. Transitions between machines power states are modelled with constant time and power overheads.

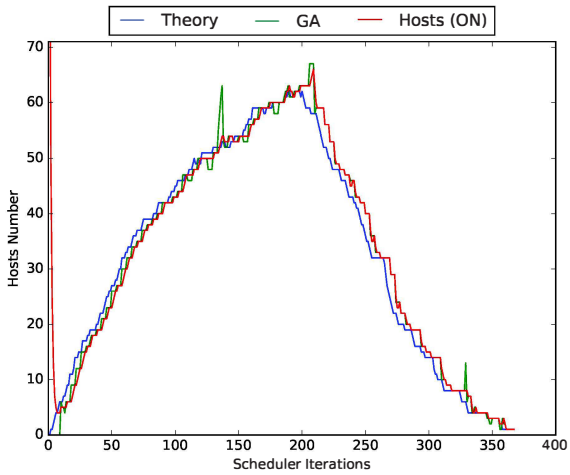
Figure 3(a) plots the energy consumption for all the algorithms on different loads. We can see that *BestFit* is almost as good as *SOPVP* for low loads, but when the load increases *SOPVP* is around 6-7% better. The main reason is that there is not much reallocation to be made at low load to increase energy performance, thus meaning that the initial allocation has the most importance. However, when the load increases reallocation improves energy efficiency. Figure 3(b) represents the average waiting time aspect of the same experiments. There is a small difference between *BestFit* and *SOPVP*. This difference is mainly due to the fact that reallocating some tasks frees up some space to allocate more rapidly the arriving

TABLE III: Variables for the algorithm.

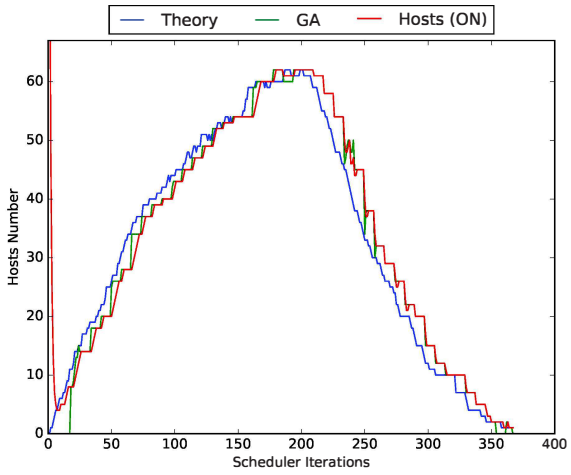
POPULATION_SIZE	Number of individual in the population
GENERATION_NUMBER	Number of generations
T_SIZE	Selection size of individuals with the tourmament
CROSSOVER_PROBABILITY	Probability of a crossover
GA_MUTATE_X_PROBABILITY	Probability of a mutation of chromosome X
GA_MUTATE_Y_PROBABILITY	Probability of a mutation of chromosome Y



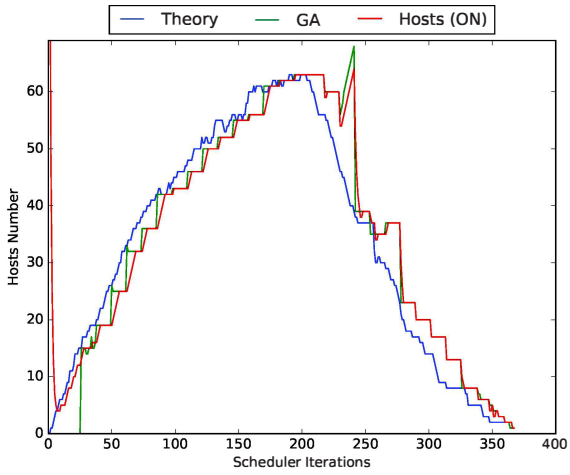
(a) Time window 4



(b) Time window 8



(c) Time window 16



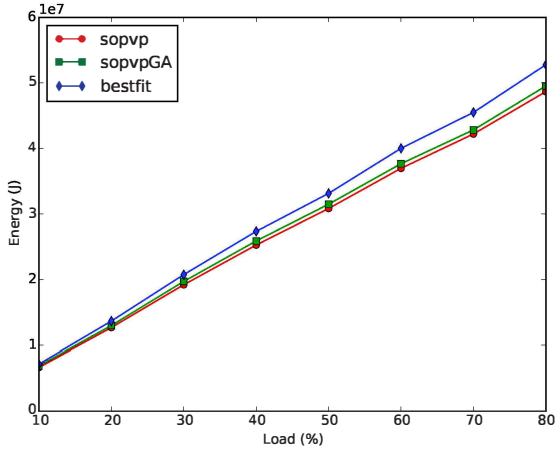
(d) Time window 24

Fig. 2: Time window comparison

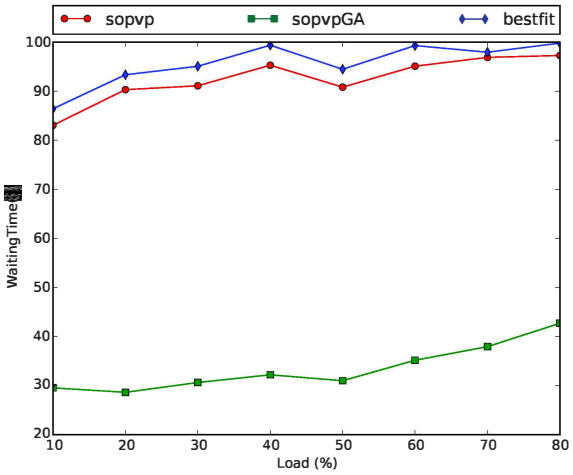
tasks. One should note that the differences between *SOPVP* and *BestFit* are only due to the reallocation, and that with the reallocation, we can gain in both energy savings, and QoS. We can see that the algorithm *SOPVPGA* is very close to *SOPVP* in term of energy saving but it is more effective than the other algorithms for the waiting time metric and thus it proposes the best quality of service for the user.

C. Host management strategies comparison

In this section, we compare the two host management policies that we propose. In the literature, in dynamic provisioning studies, future load is estimated and approximated through various ways [16]. One simple algorithm predicts all future values to be the exact same load rate as was last seen. Another one maintains the load rates seen over the last n seconds and averages the values in the window for the prediction. We compare the genetic algorithm to manage the hosts with these load prediction heuristics which we call *LastLoad* and *LastTwoLoads*. *LastLoad* is *SOPVP* algorithm with a number of hosts computed with the last mean request of tasks and the last mean capacity of hosts at the previous time step. *LastTwoLoads* works similarly but uses the mean of the two previous numbers from the last two time steps. We have chosen these two load prediction heuristics because they

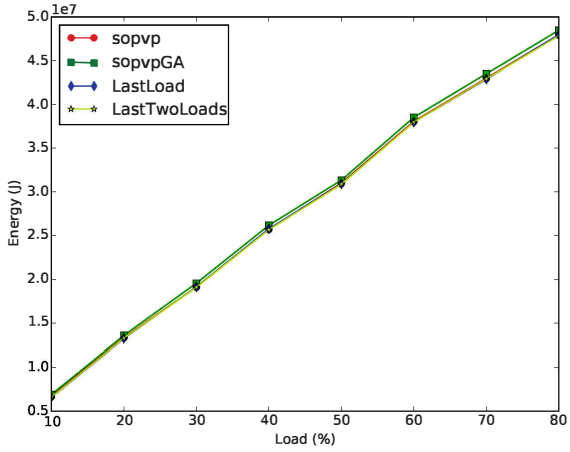


(a) Energy consumption vs System Load

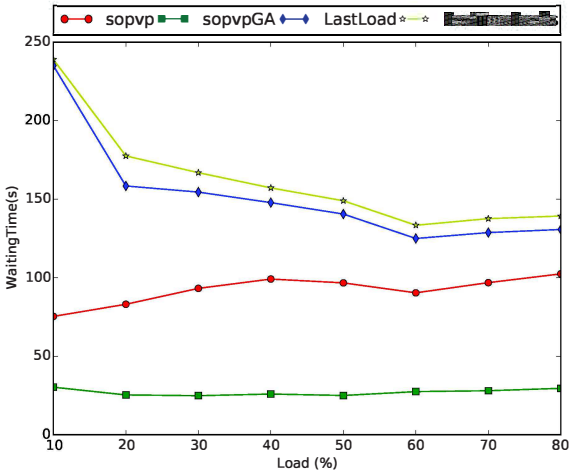


(b) Waiting times vs System Load

Fig. 3: Energy and waiting time for different loads



(a) Energy



(b) Waiting time

Fig. 4: Energy and waiting time for different loads and host management heuristics

reflect the dynamicity of the load variation in the infrastructure and are known as simple prediction heuristics. For loads between from 10% to 80% making 10 iterations for each load we compare the results of *SOPVPGA* with *LastLoad* and *LastTwoLoads*. We can see in figure 4 that the heuristics have close values for energy consumption. *SOPVP* consumes 3% less than *SOPVPGA* for a load of 10%, the difference is reduced to 1% for a load of 80%. Differences between *LastLoad*, *LastTwoLoads* and *SOPVP* are less than 1% for a load of 10% and around 0.2% for a load of 80%. *SOPVPGA* has a better waiting time than all the others. Prediction and dynamic provisioning are very complex ; simple heuristics like *LastLoad*, *LastTwoLoads* or *Pivot* can help saving energy but do not perform well for QoS metric like the waiting time. The *GA heuristic* proposed in this article performs better.

V. Related Work

In the literature, energy efficiency is addressed at different levels [5][6][17][18][19][20]. At the node, infrastructure or middleware level. Some studies propose datacenters powered at least partly with renewable energy [21] ; these solutions are often based on the over provisioning of the electrical infrastructure. Cooling cost is also addressed by improving the methods of heat extraction [22] or with mapping algorithms to avoid hot spots [23]. To reduce energy cost, at the middleware level, placement algorithms are proposed. Dynamic consolidation aims at reducing the number of hosts used and is usually based on virtual machine migrations. Migration overhead has to be taken into account as the overheads due to powering on and off a node. Entropy [24] is a resource manager which performs dynamic consolidation taking into account migration overhead. Entropy periodically optimizes the VM placement as in our approach. The difference is that we also take into account on/off overheads. In [25], the authors detect underload and overload conditions and dynamically solve them. It provides a framework for dynamic consolidation of VMs based on the OpenStack platform. In [26], they propose an energy-aware algorithm based on a pareto multi-objective approach dealing with both energy consumption and Service Level

Agreement (SLA). Other heuristics use genetic algorithms [12] to find a near optimal task placement. In [27], they present a multi-objective genetic algorithm that optimizes the energy consumption, CO2 emissions and the generated profit of a geographically distributed cloud computing infrastructure. Our contribution takes the advantages of fast mapping heuristics while retaining the quality of a genetic algorithm for host management strategy.

VI. Conclusion and Future Work

In this paper, we have studied energy efficiency at the middleware level through mapping algorithm and green leverages: server states (on/off) and migration. We have proposed a host management policy to minimize the overheads. The host management policy aims at correctly provisioning the number of machines to power on. The policy proposed is based on a genetic algorithm and is periodically used with a consolidation algorithm. We have evaluated for different loads the performance of the genetic algorithm compared to other provisionning policies and to other mapping heuristics. The performance is better and helps to reduce energy consumption while keeping a good QoS. The originality of this work is the proposal of a regression formula computed with a genetic algorithm. In future work we plan to propose new mapping algorithm, long term prediction and add other green leverages.

Acknowledgment

The work presented in this paper was partially supported by the French ANR DATAZERO project, ANR-15-CE25-0012 and the ANR SOP project ANR-11-INFR-001.

References

[1] "Green500," www.green500.org.
[2] J. Koomey, "Growth in data center electricity use 2005 to 2010," 2011.
[3] W. Heddeghem, S. Lambert, B. Lannoo, D. Colle, M. Pickavet, and D. P., "Trends in worldwide ict electricity consumption from 2007 to 2012," *Computer Communications*, vol. 50, pp. 64–76, 2014.
[4] C. Gary, "Clicking clean: How companies are creating the green internet," *Greenpeace International*, 2014.
[5] A.-C. Orgerie, M. Dias de Assuno, and L. Lefvre, "A survey on techniques for improving the energy efficiency of large scale distributed systems," *ACM Computing Surveys*, vol. 46, no. 4, April 2014.
[6] G. L. Valentini, W. Lassonde, S. U. Khan, N. Min Allah, S. A. Madani, J. Li, L. Zhang, L. Wang, N. Ghani, J. Kolodziej, H. Li, A. Y. Zomaya, C. Z. Xu, P. Balaji, A. Vishnu, F. Pinel, J. E. Pecero, D. Kliazovich, and P. Bouvry, "An overview of energy efficiency techniques in cluster computing systems," *Cluster Computing*, vol. 16, pp. 3–15, 2013.
[7] A. J. Younge, R. Henschel, J. T. Brown, G. von Laszewski, J. Qiu, and G. C. Fox, "Analysis of Virtualization Technologies for High Performance Computing Environments," in *IEEE 4th International Conference on Cloud Computing*. IEEE, 2011, pp. 9–16.
[8] A. Gupta, L. V. Kale, D. Milojicic, P. Faraboschi, and S. M. Balle, "HPC-Aware VM Placement in Infrastructure Clouds," in *IEEE International Conference on Cloud Computing*. IEEE, 2013, pp. 11–20.
[9] V. Mauch, M. Kunze, and M. Hillenbrand, "High performance cloud computing." *Future Generation Computer Systems*, vol. 29, no. 6, pp. 1408–1416, 2013.
[10] "Xlcloud project," http://xlcloud.org.
[11] D. Borgetto and P. Stolf, "An Energy Efficient Approach to Virtual Machines Management in Cloud Computing." in *International Conference on Cloud Networking, Luxembourg, 08/10/2014-10/10/2014*. http://www.ieee.org/: IEEE, 2014, pp. 229–235.
[12] L. Srinivas and L. Patnaik, "Genetic algorithms: a survey," *Computer*, vol. 27, no. 6, pp. 17–26, 1994.
[13] B. L. Miller and D. E. Goldberg, "Genetic Algorithms, Tournament Selection, and the Effects of Noise," in *Complex Systems*, vol. 9, 1995, pp. 193–212.
[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.
[15] "Standard performance evaluation corporation, specpower_ssj2008 results," http://www.spec.org/power_ssj2008/results/, October 2014.
[16] A. Krioukov, P. Mohan, S. Alspaugh, L. Keys, D. Culler, and R. Katz, "Napsac: Design and implementation of a power-proportional web cluster." *ACM SIGCOMM Computer Communication Review*, vol. 41, pp. 102–108, 2011.
[17] F. Kong and X. Liu, "A Survey on Green-Energy-Aware Power Management for Datacenters," *ACM Computing Surveys*, vol. 47, no. 2, 2015.
[18] G. L. Tsafack Chetsa, L. Lefevre, and P. Stolf, "A Three Step Blind Approach for Improving HPC Systems' Energy Performance," in *Energy Efficiency in Large Scale Distributed Systems conference (EE-LSDS), Vienna, 22/04/2013-24/04/2013*. http://www.springerlink.com: Springer, 2013, pp. 168–181.
[19] G. L. Tsafack Chetsa, L. Lefvre, J.-M. Pierson, P. Stolf, and G. Da Costa, "Exploiting performance counters to predict and improve energy performance of HPC systems," *Future Generation Computer Systems*, vol. 10.1016/j.future.2013.07.010, pp. 287–298, juillet 2014. [Online]. Available: http://oatao.univ-toulouse.fr/12657/
[20] D. Borgetto, R. Chakode, B. Depardon, C. Eichler, J. Garcia, H. Hbaieb, T. Monteil, E. Pelorce, A. Rachdi, A. Al Sheikh, and P. Stolf, "Hybrid approach for energy aware management of multi-cloud architecture integrating user machines," *Journal of Grid Computing, special issue on Green Cloud Computing*, vol. 10.1007/s10723-015-9342-y, 2015.
[21] "How clean is your cloud ?" http://www.greenpeace.org.
[22] "Free cooling," www.ecoinfo.cnrs.fr/article140.html.
[23] H. Sun, P. Stolf, J.-M. Pierson, and G. Da Costa, "Energy-efficient and thermal-aware resource management for heterogeneous datacenters," *Sustainable Computing: Informatics and Systems*, vol. 10.1016/j.suscom.2014.08.005, pp. 1–15, aot 2014.
[24] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. L. Lawall, "Entropy: a Consolidation Manager for Clusters," in *VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. New York, NY, United States: ACM, 2009, pp. 41–50.
[25] A. Beloglazov and R. Buyya, "Openstack neat: a framework for dynamic and energy-efficient consolidation of virtual machines in openstack clouds." *Concurrency and Computation: Practice and Experience*, vol. 27, pp. 1310–1333, 2015.
[26] Y. Kessaci, N. Melab, and E. G. Talbi, "A multi-start local search heuristic for an energy efficient vms assignment on top of the opennebula cloud manager." *Future Generation Computer Systems*, vol. 36, pp. 237–256, 2014.
[27] ——, "A pareto-based metaheuristic for scheduling hpc applications on a geographically distributed cloud federation." *Cluster Computing*, vol. 16, pp. 451–468, 2013.