A Multi-start Local Search Heuristic for an Energy Efficient VMs Assignment on top of the OpenNebula Cloud Manager

Yacine Kessaci, Melab Nouredine, El-Ghazali Talbi

A Multi-start Local Search Heuristic for an Energy Efficient VMs Assignment on top of the OpenNebula Cloud Manager

Y.Kessaci, N.Melab
and E-G.Talbi

INRIA Lille, CNRS/LIFL, Université Lille 1.
Parc Scientifique de la Haute Borne,
6 rue Héloise Bât.B, Park Plaza,
59650 Villeneuve d'Ascq, France.
Email: {yacine.kessaci, nouredine.melab, talbi}@lifl.fr

Abstract

Reducing energy consumption is an increasingly important issue in cloud computing, more specifically when dealing with a large scale cloud. Minimizing energy consumption can significantly reduce the amount of energy bills, and the greenhouse gas emissions. Therefore, many researches are carried out to develop new methods in order to consume less energy. In this paper, we present an Energy-aware Multi-start Local Search algorithm (EMLS-ONC) that optimizes the energy consumption of an OpenNebula based Cloud. Moreover, we propose a Pareto Multi-Objective version of the EMLS-ONC called EMLS-ONC-MO dealing with both the energy consumption and the Service Level Agreement (SLA). The objective is to find a Pareto tradeoff between reducing the energy consumption of the cloud while preserving the performance of Virtual Machines (VMs). The different schedulers have been experimented using different arrival scenarios of VMs and different hardware configurations (artificial and real). The results show that EMLS-ONC and EMLS-ONC-MO outperform the other energy- and performance-aware algorithms in addition to the one provided in OpenNebula by a significant margin on the considered criteria. Besides, EMLS-ONC and EMLS-ONC-MO are proved to be able to assign at least as many VMs as the other algorithms.

Keywords:
Energy-aware scheduling, cloud computing, resource allocation, OpenNebula, cloud manager, multi-start local search.

1. Introduction

Cloud computing is an emerging computer science paradigm of distributed computing in which applications, data and infrastructures are proposed as a service that can be consumed in a ubiquitous, flexible and transparent way. Cloud computing brings with it such benefits via cloud managers which hide to the user some complex and challenging issues such as scheduling. However, the solutions to these issues provided in cloud managers are often limited. For instance, the scheduling approach proposed in many cloud managers like OpenNebula is limited regarding the criteria taken into account. Despite the increasing impact of the energy on many applications

[1], cloud managers are still rarely designed around the energy consumption reduction concept. In this paper, we address energy- and performance-aware scheduling of different VM configurations over cloud infrastructures. We propose a multi-start parallel local search heuristic for cloud managers especially for OpenNebula.

According to an Amazon's estimate [2], the energy-related costs amount represents 42% of the total data center budget, and includes both direct power consumption 19% and cooling infrastructure 23%, these values are normalized with a 15 years amortization. It clearly appears that minimizing energy consumption is an important and challenging issue to deal with.

Some works [3, 4] have been conducted for energy reduction purposes over the previous hardware architectures. However, the cloud transition and the generalization of the usage of cloud managers, let the integration of the previous approaches difficult. Indeed, the new optimization approaches have to be as seamless as possible, which was not the case of the previous ones. The reason, is the exclusivity that relates these techniques to their model. They are in general made for specific tasks and specific hardware (DVFS).

Furthermore, the evolutions brought by the cloud managers skipped the upgrade process of the energy-aware techniques for this new concept. Indeed, among the cited opportunities offered by cloud managers, only very few ones are energy-aware based or propose options conducting to energy savings. Moreover, to the best of our knowledge, none of the existing approaches use a powerful technique such as metaheuristic for scheduling. Besides, none of the existing real-world clouds, managed by OpenNebula or not, take natively into account energy-saving concerns or any other dealt with criteria in their assignment policies. Therefore, the general objective of our work is to deal with all these lacks.

In this paper, we present new approaches that tackle the energy consumption and the VMs performance issues within a realistic cloud infrastructure. We chose as a case study to use OpenNebula as a software management solution. Thus, we propose a new scheduler embedded in OpenNebula.

A virtual machine (VM) is a software based machine emulation technique to provide a desirable, on demand computing environment for users. Our scheduler is based on a multi-start local search metaheuristic that provides a better scheduling by dispatching the incoming virtual machines (VMs) according to the minimum energy consumption. We also propose a Pareto bi-objective version of this scheduler that addresses both the energy consumption criterion and the performance of the VMs.

Both algorithms that we propose use information related to each host provided by the hypervisor to find the best VM assignment. Our approaches aim to reach the previous cited objectives while assigning the maximum number of VMs. The multi-start metaheuristic [5] allows the exploration of a huge number of potential solutions of the problem. This maximizes the chances to discover the best ones. The main contributions of this paper are the following:

- An energy reduction scheduler (EMLS-ONC) embedded in the OpenNebula cloud manager.
- New energy and VMs performance evaluation models for VM assignment on cloud.
- An experimental study of the different assignment scenarios between the default scheduler provided in OpenNebula, other energy-aware based approach and the EMLS-ONC algorithm.
- A Pareto bi-objective version of EMLS-ONC (EMLS-ONC-MO) dealing with both energy and VMs performance.

- An experimental study of a real deployment of EMLS-ONC over 200 physical machines of the GRID'5000 grid infrastructure.

The remainder of the paper is organized as follows. In Section 2 we present the related work. Section 3 presents the application, system, energy and performance models investigated in our problem modeling. Our approaches are presented in Section 4. The results of our experimental study are discussed in Section 5. The conclusion is drawn in Section 6.

2. Related Work

After a race to performance, utility and cloud computing paradigms are facing the energy issue. To reduce energy consumption, various issues such as resource management at both software and hardware levels must be addressed. Software approaches are mainly based on virtualization or task consolidation. Virtualization consists in running on a single computer multiple operating systems as if they are running on separate computers or merging several physical computers to form a single virtual one. Hardware approaches use the opportunity offered by manufacturers in modern processors to adjust the voltage and frequency. This adjustment may consist in varying the processor performance and thus its energy consumption.

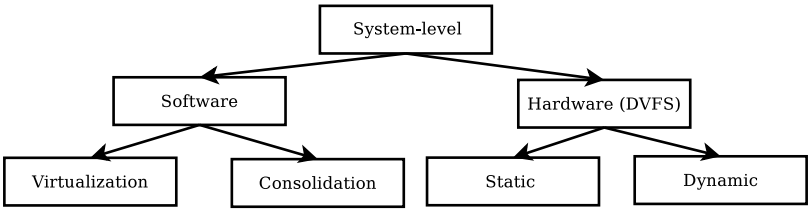

Figure 1: Classification of system-level energy reduction techniques

Therefore, several works have been proposed in the field of energy-aware computing. The proposed approaches are based either on software or hardware layer of the system (i.e. system-level) (see Figure 1). Most of the energy-aware approaches tackle this issue by referring and focusing on scheduling dedicated applications. Those methods belong to the hardware branch of the system-level tree. In [6, 3], a hardware technique (DVFS) consisting in varying the CPU frequency in order to minimize the energy consumption is proposed. The drawback of this type of methods is the assumption on the existence of a tight coupling between tasks and resources.

Regarding the software methods, the task consolidation is an effective method to increase resource utilization for energy consumption reduction purposes. Task consolidation is the process of assigning a set N of n tasks (service requests or simply services) to a set R of r resources without violating time constraints. This technique aims to maximize resource utilization while minimizing energy consumption. Indeed, a resource allocation strategy that takes into account resource utilization should lead to better energy efficiency.

The paper [7] exposes some of the complexities in performing consolidation for power optimization, and proposes some research directions to address the involved challenges. In [4], the authors present two energy-conscious task consolidation heuristics ECTC and MaxUtil. These two heuristics aim to maximize resource utilization. They explicitly take into account the energy consumption of both used and idle processors. The proposed heuristics assign each task to

the resource on which the required energy consumption for executing the task, is explicitly or implicitly minimized without performance degradation of that task.

Another way to reduce cloud computing energy footprints is proposed in [8]. The authors present a reinforcement learning approach to deal with the optimization of two main criteria: performance and power consumption. All the previous presented works aim to reduce the energy consumption in single data centers or in geographically concentrated multiple servers except the works proposed in [9] and in [10]. In [9] the author deals with energy consumption reduction in large-scale computational grids like Grid'5000 by switching off idle nodes in a clever way, while the work in [10] deals with a geographically distributed cloud federation, where a Pareto multi-objective genetic algorithm is proposed to address the energy consumption, gas emissions and pricing issues.

Another type of works models the assignment problem of VMs as a bin packing where the VMs are the objects to pack and the machines the boxes. [11] proposes an architecture called pMapper that helps to assign applications taking into account both the power and performance cost. The algorithm is based on a First Fit Decreasing (FFD) algorithm. FFD and its variants have been proved in a comparing study [12] to perform well among the other heuristics for VM assignment. However, the major issue with this type of algorithms is the lack of diversity. Indeed, they provide the global optimum (best solution) only for landscapes with a single local optimum. However, the scheduling problem changes according to the different VM arrivals. Thus, significant modifications of the solution landscape may occur creating different local optima. It is then necessary to explore other local optima using metaheuristics such as the approach that we propose in this paper to enhance the scheduling.

Lately, a significant evolution in the field of parallel and distributed computing led to a massive usage of clouds. Therefore, virtualization and consolidation techniques have been integrated to this new model through middlewares called cloud managers, such as Openstack [13], OpenNebula [14], Eucalyptus [15], Nimbus [16], etc.

The problem is that the major works including cloud managers, address only cloud broking on a three-tier model (client, broker, provider), where the objective is to optimize the needs of both the broker and the client. The contributions are focused on the VM placement strategies for the broker. Indeed, they do not give interest to the assignment process that happens within the cloud infrastructure on the side of the provider. The works in [17] and [18] give a good illustration of that point. In [17], the authors present a study where they compare the VM placement mechanisms over a multi-provider and multi-site cloud. They prove that a multi-cloud deployment gives better performances and minimizes the costs. However, energy issues are not addressed in this paper. The same goes for [18] where the focus is put on different scheduling strategies for an optimal deployment across multiple clouds and none of the optimization criteria concerns the energy savings.

Therefore, these works should be extended to take into account energy reduction in cloud managers. Snooze [19] is the first cloud manager that includes an energy criterion. The virtual machines that compose the cloud have the ability to switch off when they are idle. However, very few works like [20] propose an approach that assigns and schedules the VMs according to this criterion. In [20], the VM scheduling algorithm is based on the DVFS technique to reduce the energy consumption of a single OpenNebula virtualized cluster. The idea behind this work is to reduce the clock frequency of the cluster as low as possible to fit exactly the VMs requests.

The major lack of the approach proposed in [20] is that it deals only with one cluster and not with a large number of machines that compose a cloud in general. In addition, this work makes an assumption on the hardware configuration of the machines that compose the cloud, assuming

that they are equipped with DVFS.

To deal with all the misses mentioned before, we propose a scheduler for cloud managers. We chose OpenNebula because it is one of the best known cloud managers. The scheduler uses a multi-start local search (EMLS-ONC) to optimize the energy on a distributed cloud infrastructure managed by OpenNebula. In our approach we integrated our scheduler in OpenNebula taking place of its default one. This was made possible, since the software is open source. In addition, because of the integration, our approach can be deployed at the same time with OpenNebula on a real distributed infrastructure to schedule VMs. It is also important to take into account the performances of the assigned VMs as done in [11]. Indeed, only few works dealing with energy focus on the impact of this reduction on the VM performances. Therefore, we also propose in this work a Pareto multi-objective version of EMLS-ONC called EMLS-ONC-MO that deals with both the energy consumption of the infrastructure and the VM performances.

3. Distributed Cloud Scheduling Model

3.1. System Model

Our model is based on an Infrastructure As A Service (IAAS) cloud model managed by OpenNebula. We deal with a two-tier architecture with in each side respectively a distributed cloud provider and clients. The clients have a direct access to the cloud resources by requesting them from the provider. The service proposed by the cloud provider in our approach offers VMs to the clients in order to run their applications. The role of our approach is to help the provider to optimize two criteria in his/her cloud management while proposing its services.

The cloud considered in our model can very from few installed hosts to a multi-cluster distributed cloud. The goal of this approach is to find the best assignment of the VMs on the hosts which compose the cloud. We use for this a multi-start local search metaheuristic. The objective is to assign a maximum number of VMs while optimizing two criteria: energy consumption and performance of the assigned VMs. The location of the default OpenNebula's scheduler and its replacement by our EMLS-ONC (or EMLS-ONC-MO) scheduler is shown in Figure 2.

The optimization of the criteria is due to the diversity offered by the heterogeneity of the hosts that compose the cloud. The heterogeneity means different CPU, memory and storage capacities. It means also different CPU frequencies and different CPU usage on each host. This offers multiple assignment possibilities which contribute to a reduction in energy and a gain in performance.

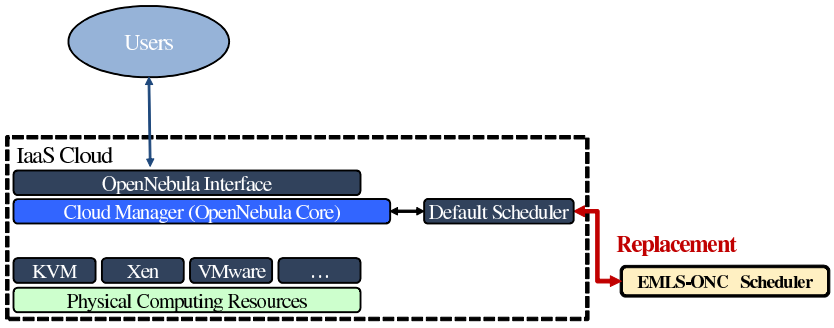

Figure 2: Overall architecture of Opennebula including the EMLS-ONC (or EMLS-ONC-MO) scheduler location

3.2. Energy Model

The energy consumption of a cloud results from IT equipments (network, storage and computing) and from auxiliary equipments (lighting, cooling ...). Our approach is computation oriented, the most of the energy consumption results then from the intensive computation. Thus, in our work we do not consider lighting equipments since their impact on the energy consumption is negligible. However, the impact of the cooling energy consumption is significant and is directly related to the computing energy consumption. We also do not use DVFS in our model to save energy. In other words, our approach does not pay attention on how the energy is optimized within the processor itself. Our scheduler is designed to be as seamless as possible to fit all the processor infrastructure with and without the DVFS feature. Our scheduler aims to prove the advantage of the hardware heterogeneity offered by the clouds in the energy reduction.

Our processor energy model is derived from the power consumption model in Complementary Metal-Oxide Semiconductor (CMOS) [21, 22]. The power consumption P of a CMOS-based microprocessor is defined in Equation (1).

$$P = ACV^2 f + I_{leak}V + P_{short} \tag{1}$$

where A is the number of switches per clock cycle, C is the total capacitance load, V is the supply voltage, f is the frequency, I_{leak} is the leakage current and P_{short} is the power dissipation resulted from switching from a voltage to another in case of a DVFS activation. This latter is not influent in our study as we do not use the DVFS method.

We notice that A and C are constant values we can then name them α (a constants product is a constant). The second part of the equation represents the static consumption, this value is also constant, we name it β. In addition, in CMOS processors the voltage can be expressed as a linear function of frequency [23, 24], $V^2 f$ is replaced by f^3. The equation becomes Equation (2).

$$P = \alpha f^3 + \beta \tag{2}$$

The main assumption of our energy model is based on the following observation: the highest the CPU usage is (see Figure 3), the hottest the CPU's temperature is (see Figure 5) and faster the cooling fan turns (see Figure 4). In all the figures, the blue and the red lines designate the information about the devices usage and temperature respectively with and without using the cool'n'quiet technology [25]. As result of this observation we notice that the CPU usage is a major parameter, and teaches a lot about the system energy behavior. In addition, our model can handle up to 100 VMs in each scheduling cycle (see Section 4.5), this can quickly become CPU intensive for the hosts of the cloud. Indeed, a loaded host means a higher CPU usage, which changes the temperature of the host's devices and in that way the cooling system behavior. Therefore, the energy needed for the VMs computation is only a part of the total energy consumption. In this work, we use only the CPU energy to make decisions about the VM assignments. As a consequence, we should have more significant energy savings if we take the auxiliary energy consumption into account. However, the objective of this approach is more to show the impact of both the heterogeneity in the hosts' frequencies and the CPU usage parameter in the objective function of the EMLS-ONC algorithm, than to rise the energy saving improvement results. The scheduler retrieves the frequency of the processors of each host and its current CPU usage by requesting the hypervisor. The used hypervisor in our model is KVM and the objective function of the energy criterion is defined in Equation (3).

$$(E)_{ij} = (\alpha_i f_i^3 + \beta_i) \times e_j \times n_j \times (\frac{CPU\_usage_i}{const\_value} + 1) \tag{3}$$

Where f_i is the frequency of the host i, e_j is the time reservation of the VM, n_j is the number of processors required for this VM and CPU_usage_i is the current CPU usage of the host i. The energy consumption is related to both the host and the VM. The *const_value* is a constant value that represents the coefficient of the energy increase according to the CPU usage increase. In other words, *const_value* represents the ratio between the CPU usage increase from the idle point to the max usage point and its effect on the energy consumption. We conducted an experiment (see Figure 6) to find the value of *const_value*. We used a wattmeter on Dell precision T7400 with an Intel Xeon X5410 2.33 GHz four processor cores. We used the Intel Xeon architecture because it is commonly used in data centers and designed for CPU intensive usage. We stressed the processor several times during 2 minutes and we found out that the extra power needed compared to the idle state, varies between 30% and 40% of the total energy consumption. We decided then that in our objective function the *const_value* equals 3, which represents 33% of extra power between the idle and the maximum processor usage.

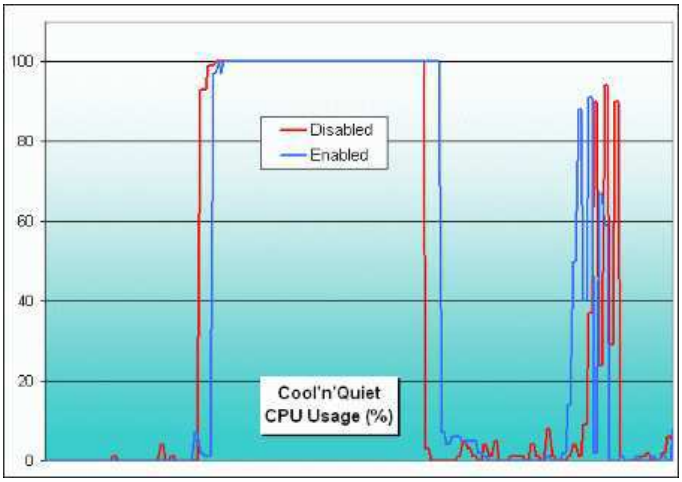

Figure 3: CPU usage during a stress period (x-axis)

3.3. VM Performance Model

Reducing the energy consumption in a cloud infrastructure is a major issue. However this could conduct to some drawbacks in terms of performance. The virtualization tool offered by the cloud allows different VMs to share the same physical host. The energy consumption reduction conducts often to gather the VMs into the same physical host. However, to take benefit from those VMs and from their potential, a total isolation between the different VMs has to be provided by the hypervisor within the same host.

The CPU resources do not cause problems and usually respond well to the isolation. However, the cache memory aspect is more tricky to handle. Indeed, sharing a physical resource means sharing CPU cache memory as well. The problem is that there is not as much cores as caches to keep a total isolation between the VMs. This problem is not significant for the VMs with low memory needs. However, when the VMs needs exceed the capacity of the L2 cache size, the VMs are not isolated any more [11]. In [11], the experiments on the VM isolation show that for the VMs with high memory needs, the response time (delays) of the VMs increases

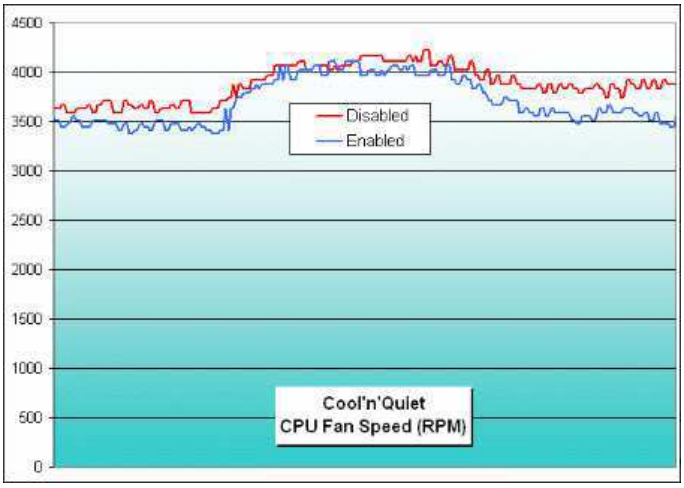

Figure 4: FAN rpm during a stress period (x-axis)

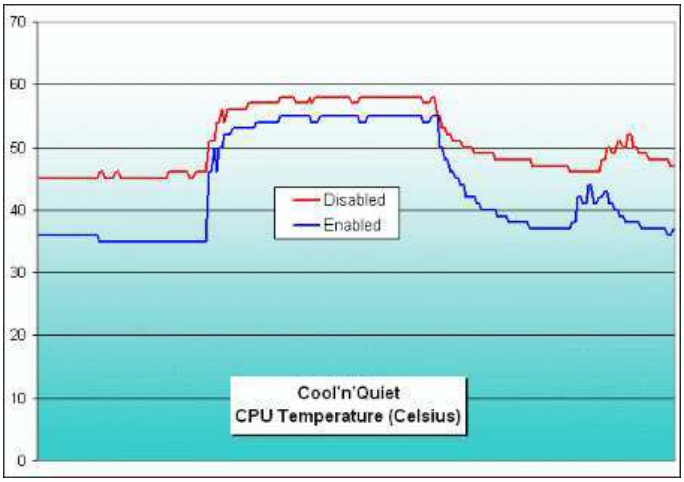

Figure 5: CPU temperature during a stress period (x-axis)

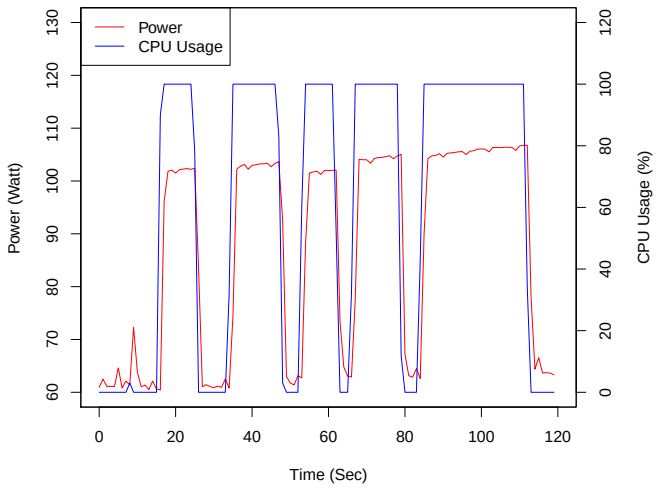

Figure 6: CPU usage vs energy consumption

due to the cache misses. We noticed that the response time of the VMs double between the value 0% of back ground utilization (lonely VM) and 100% of back ground utilization (full physical host). We notice also that the increase of the response time is linear with the memory increase. We deducted then the Equation (4) from this behavior to link the VM response time (VM performance) $Response\_time_{ij}$ to the VM memory needs and the host memory usage.

$$Response\_time_{ij} = Memory_j + Memory_j \times Mem\_usage_i \quad (4)$$

Where, $Memory_j$ is the amount of memory needed by the VM j and Mem_usage_i is the current memory usage of the host i.

3.4. Problem Description

As illustrated in Figure 7, the focus in our work is on a two-tier cloud model. The first tier is a cloud provider which has N heterogenous hosts (data centers). The second tier is a set of clients with J VMs requests for running their applications. The problem consists of scheduling J VMs on N data centers. We know that the task scheduling problem in general is NP-hard [26]. Therefore, our VMs scheduling problem is NP-hard as well. Thus, a metaheuristic algorithm appears to be the most appropriate approach to solve the problem. The metaheuristic that we used is a multi-start local search. The multi-start part is used to bring diversification in the problem solving by a bigger exploration, while each local search adds accuracy with the intensification. Our approach provides both diversification and intensification without the processing time drawback of an evolutionary approach. In other words, EMLS-ONC (or EMLS-ONC-MO) returns the assignment within the time limit (*scheduling cycle* in Section 4.5).

With OpenNebula, the client submits virtual machine requests with requirements. Those requirements are the number of CPU, memory size, storage capacity, the type of the operating

system, etc for the VM. In our problem, we added a time requirement in the definition of the VM to know the duration of the execution and to get an estimation of the energy consumption.

During the scheduling process, the user submits a request for a VM j. A VM in our model is defined by a triplet (e_j, n_j, m_j), all the triplet information are given by the user during the submission, except the starting time of the VM (t_j) which is deduced from the submission time. The elements of the triplet represent the duration reservation time of the VM (e_j), the number of processors needed by the user for his/her VM (n_j) and finally the memory size (m_j). Our triplet is inspired from Amazon EC2 [27] except for the reservation time. This parameter is compulsory in our model in order to compute the energy consumption. The time unit of the reservation time is one hour. Thus, the user has sometimes to reserve his/her VM for a time longer than needed to ensure the completion of the application that he/she runs on it.

The objective function of our approach aims at minimizing the energy consumption of the entire infrastructure when hosting the VMs (EMLS-ONC). A second objective function is used in addition to the previous one to maximize the performance of the VMs in EMLS-ONC-MO. They are formulated in Equations (5) and (6):

$$Minimizing\ the\ energy\ consumption = \sum_{i}^{N}\sum_{j}^{J}(E)_{ij} \tag{5}$$

Where $(E)_{ij}$ is the power consumption of the host i while executing the VM j. This is always done by respecting the following constraints:

- Each VM j has to find at least one host with the correct requirements to be assigned on it, otherwise the VM is rejected.
- Each VM j can be assigned to one and only one host i.

$$Maximizing\ the\ VMs\ performance = Minimizing(\sum_{i}^{N}\sum_{j}^{J} Response\_time_{ij}) \tag{6}$$

Where $Response\_time_{ij}$ is the response time of the VM j while assigned to the host i including the delays. Note that, the VM performance is inversely proportional to the response time of the VM.

The two objectives in the multi-objective version (EMLS-ONC-MO) are tackled in a Pareto way, while assigning the maximum number of VMs is a priority. In other words, the best found solution has first to be the one that assigns the highest number of VMs. After that, a Pareto ranking is done to classify the solutions. The final best solution is the assignment that maximizes the number of VMs with the best energy consumption for EMLS-ONC or the best Pareto value for EMLS-ONC-MO.

4. Multi-start Local Search Algorithm for VMs Assignment

Before describing our EMLS-ONC and EMLS-ONC-MO approaches, we describe briefly the following sections how the default OpenNebula scheduler and bin packing FFD-based algorithm work.

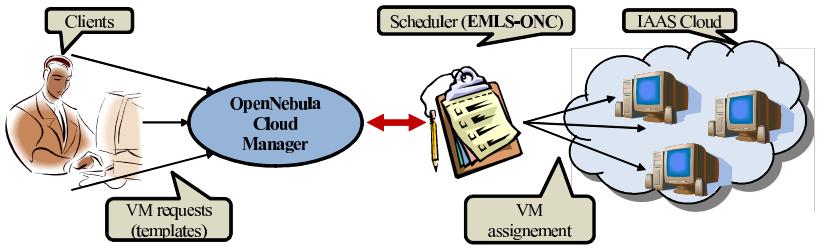

Figure 7: Two-tiers cloud model representing the client tier, the provider tier (Cloud), and the relationship between both OpenNebula and EMLS-ONC (or EMLS-ONC-MO) scheduler.

4.1. OpenNebula Scheduler Algorithm

To make the assignment of the arriving VMs, it iterates over the set of the arrived VMs. For each VM it checks the set of hosts where it could be assigned. The first host that satisfies the VM requirements is chosen to run the VM. The scheduler stops when no more pending VMs remain. This approach is similar to the consolidation technique which is used for energy reduction purposes.

4.2. Bin packing FFD-based Scheduler Algorithm

This approach is the second one to which we compare the EMLS-ONC and EMLS-ONC-MO. It is based on the First Fit Decease heuristic (FFD) as the work [11]. As said previously, the algorithms that were based on FFD bin packing have been proven to perform the best on the energy reduction [12]. The idea is to sort the pool of the VMs to be assigned in a decreasing order (i.e. from the most to the least requirement needs) and to assign those VMs each time to the server with the least energy consumption according to Equation (2). This algorithm is the energy-aware version that has been compared to EMLS-ONC. A memory-aware version of the bin packing FFD scheduler has also has been implemented. This version assigns the sorted VMs to the server with the most important memory capacity. This helps to avoid a memory overload which will lead to cache misses and therefore to decrease in the VMs performance. The Pareto EMLS-ONC-MO has been compared to both energy- and memory-aware bin packing FFD algorithms.

4.3. Problem Encoding

In order to formulate our problem without overriding the previous constraints (i.e. each VM has to find a host with its requirements and can be scheduled only on one host), we propose an encoding for both EMLS-ONC and EMLS-ONC-MO solutions (see Figure 8).

Figure 8 represents one possible assignment among plenty proposed by the multi-start local search algorithm. In the proposed example we identify three major specifications. The values of the first row of the table (map keys) depict the VMs that are assigned, the number which is contained by each cell of the second row of the table identifies the host to which the VM is allocated. In other words, in Figure 8, the first column represents the first VM of the pool that is currently treated by EMLS-ONC (or EMLS-ONC-MO), it is identified with the id 1 and is assigned to the host 5. The second VM with the id 3 is assigned to the host 0 and so on. This encoding includes the number of VMs contained in the pool (10 VMs in our example). It also helps one to deal with the characteristics of our problem. Indeed, it allows the processing of all

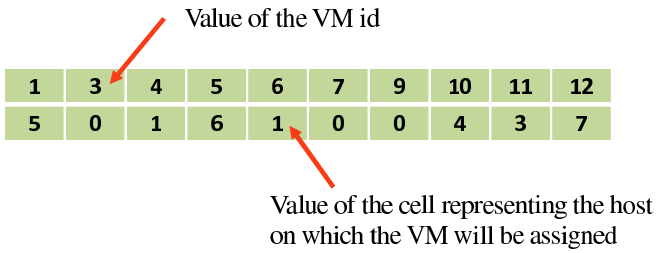

Figure 8: Problem encoding.

the VMs of the pool. Each VM will be assigned to one and only one host (no duplication of mapping keys). A host can handle more than one VM and not all the hosts are necessarily used in each solution.

4.4. Solution Initialization

The generation of the initial solution in a local search algorithm is an important phase. In fact, this step affects quality of the future results. In our approach, since we deal with a multi-start method, each local search execution has its own initialization and then its proper initial solution. This process follows 3 different methods. Just after the common phase of the hosts filtering (removing the unusable hosts with bad requirements), each VM obtains a set of hosts on which it can be assigned. The first method assigns the VM to the first available host from its set of hosts. The second method assigns the VM to the best energy efficient host available in its set of host. The last method assigns the VM to a random host in its set of hosts. The first two methods are respectively for the first two local searches of the multi-start. The last one is for the rest of the local searches to add diversity. For each host selection, a checking mechanism is applied to verify if the constraints and the availability of the host are satisfied. Indeed, during the initialization some *a priori* available host can become unavailable in case where previous VMs in the current initialization already used the resources. If no hosts are available for the VM, the VM is removed from the current scheduled pool and will be considered as failed.

4.5. Scheduling Steps

The EMLS-ONC and EMLS-ONC-MO approaches are metaheuristics-based schedulers. EMLS-ONC is a multi-start method that launches a set of local searches in order to find the best energy-aware VM assignment over the cloud. The Pareto multi-objective version EMLS-ONC-MO adds the VM performance optimization to its criteria. Before each scheduling, the EMLS-ONC or EMLS-ONC-MO scheduler waits for a fixed period of time called *scheduling cycle*. This period allows one to gather a pool of VMs in order to have a larger choice in the assignment and thus to optimize the future assignments. Once this phase done the pool of hosts is filtered out to keep only the hosts with the correct requirements. The multi-start phase as said before launches each local search algorithm separately. The number of launched LS is equal to the minimum value between the number of hosts composing the distributed cloud and 20. This parameter choice is due to the relationship between the complexity of the problem and the number of hosts. Indeed, a small number of hosts makes easier the assignment since it reduces the possibilities of VMs assignments. Therefore, few local searches are enough to get a good solution. However, the drawback of relating the number of launched local searches to the number of hosts is the processing time. A tradeoff has been found by bounding their number to 20. After the end of each local

Algorithm 1 Local Search Pareto archive ranking

```
Input: LocalSearch(n) with InitialSolution
  for each newSolution do
    if newSolution is not dominated by any solution of the archive then
      add newSolution to archive;
    end if
    for each solution in archive do
      if solution is dominated by newSolution then
        delete solution from archive;
      end if
    end for
  end for
Output: A set of Pareto non-dominated solutions
```

search process, all the best solutions of each LS are compared. Only the best solution among all the LSs solutions is kept and chosen to be the used assignment. In the last step, OpenNebula dispatches the VMs according to this assignment, it updates the hosts states and a new scheduling cycle is started. Figure 9 draws the different steps of both the EMLS-ONC and EMLS-ONC-MO. However, there are differences in the meaning of each step between the two approaches in the flowchart. The local searches in the EMLS-ONC-MO are obviously multi-objective, while in the EMLS-ONC they are not. Therefore, in EMLS-ONC-MO each LS obtains after its process a set of Pareto non-dominated solutions that are sorted in a private archive (see Algorithm 1). After the end of all the processes of all the LSs, each local search updates a common archive with the elements of their private archive to have the final archive of non-dominated solutions using Algorithm1. The penultimate step of the flowchart (Best scheduling) consists in the Pareto version of our approach to pick up randomly in the common archive a solution that will be the selected assignment. This is made possible by the equivalence of the solutions (non-dominated). In the single objective version EMLS-ONC, the *Best scheduling* flowchart step represents the best energy efficient solution among the solutions of each local search.

4.6. EMLS-ONC/EMLS-ONC-MO Algorithm

The role of the local search algorithm is to generate a number of combinations from the initial solution using neighborhood operators in order to find the best assignment according to the specified objective(s). The multi-start adds diversity, while each local search provides intensification. The local search algorithm starts by generating the initial solution. The initialization process is explained in Section 4.4. This initial solution is used to generate a neighborhood based on two neighborhood operators. The use of one or the other depends on the size of the cloud and the number of VMs. Both operators are based on an exchange process. The first operator is dedicated to generate neighborhoods for small cloud configuration or small VM number (no VM limit and less than 50 hosts or less than 5 VMs and no host limit), while the second is dedicated to large neighborhoods with large clouds and number of VMs. The first operator (see Figure 10) switches the value of the host of each VM of the initial solution with each value of the VM's hosts set exhaustively. In other words, the neighborhood operator checks all the VM's available hosts to find the one where the VM consumes the less energy for the EMLS-ONC or the best Pareto solution for EMLS-ONC-MO. In the second operator, the number of both hosts and VMs

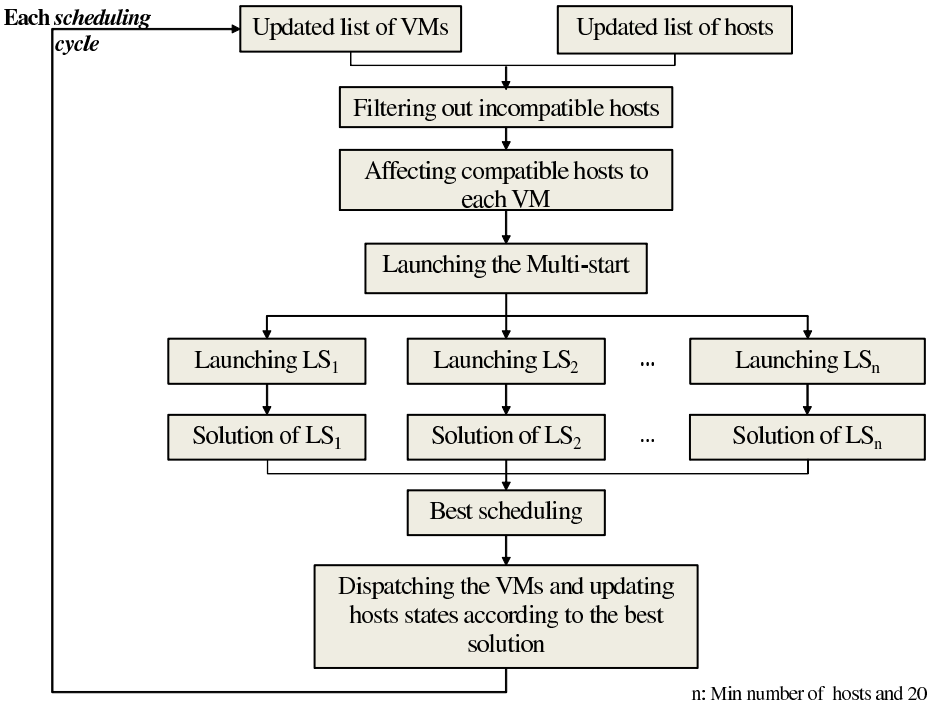

Figure 9: The Flowchart of the EMLS-ONC algorithm.

is bigger. Therefore, the algorithm can not afford to enumerate all the combinations in a reasonable time. Thus, it switches the selected host with not all the VM's hosts set but only with a randomly selected range of hosts among this set. Therefore, one iteration of each LS generates one neighborhood. The local search ends when it enumerates all the hosts of the hosts set of each VM with the first operator, or enumerates all the hosts of a selected range in the hosts set of each VM for the second operator. Each solution of the generated neighborhood is checked for its feasibility. A fitness value is also assigned to this solution. The best solution of the neighborhood is kept to build another neighborhood during the next iteration using the previous operators. The algorithm stops when the number of iterations in each LS reaches the number of VMs. As for the number of local searches in the multi-start, the reason of choosing this value is due to the complexity of the problem. Indeed, the more VMs they are the more iterations are needed to find a good solution.

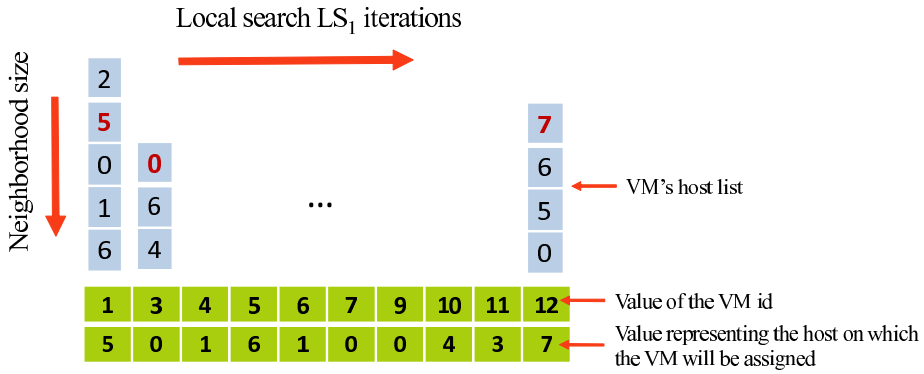

Figure 10: The Mechanism for generating the neighborhood from an initial solution (neighborhood operator) of one of the local search algorithm that composes the EMLS-ONC/EMLS-ONC-MO scheduler.

5. Experiments and Results

This section presents the results obtained from our comparative experimental study. The experiments aim to demonstrate and evaluate the contribution of the EMLS-ONC and EMLS-ONC-MO approaches. The comparison is done on the different addressed criteria with an energy- and memory-aware approaches and the default OpenNebula scheduler.

5.1. Experimental Settings

The experimental settings concern both sides of our two tier cloud model. The client side with the VMs features and the provider side with the hardware configuration of the cloud.

5.1.1. Artificial Hardware Settings

- **VMs' settings:** Concerning the inputs of the scheduler we generated VMs in an XML format. Thus, the OpenNebula parser reads their features in a realistic way. The VM features in our experiments vary according to three parameters. Indeed, as said before in Section 3.4 with the triplet (e, n, m), the VMs parameters are the execution time, the number of processors and finally the memory needs. In order to fit the algorithm parameters,

we generated randomly this triplet where the execution time e is a value from [1, 10] hours, the processor requirement n varies in the interval [0.5, 9] and the memory needs m in [1, 8] GBs for the CPU-intensive VMs and [1, 15] GBs for the memory-intensive VMs. The subdivision unit of all the intervals is given in Equations (7), (8) and (9).

$$e = 1 + k; 0 \leq k \leq 9 \tag{7}$$

$$n = k/2; 1 \leq k \leq 18 \tag{8}$$

$$m = 1 + k; 0 \leq k \leq 14 \tag{9}$$

- **Distributed cloud settings:** As for the VMs the hosts features are provided in XML format to the OpenNebula parser. Hence, we generated different types of hosts by changing each time their features. Each host is specified by its number of cores randomly generated between [1, 24] cores, its memory capacity from [2, 24] GB and its CPU frequency from [1, 3] Ghz. A host has also information about the amount of CPU and memory used resources, those values have to never exceed the initial capacity of the host for both CPU and memory. The value of the free resources for each device (CPU, memory) is deduced from the initial host capacity minus the amount of the current used resource. The subdivision unit of all the hosts' parameters intervals is 1. In other words, all the intervals are composed of integer values.

The interval values of the VM features presented above are deduced from the type of VMs proposed by the cloud providers like EC2 for Amazon [27]. The instances of EC2 vary from small one (1 CPU, 1.7 GBs memory) to extra large one (8 CPUs, 15 GBs memory) [28]. The floating values as 0.5 for the VM CPU needs are due to virtualization that allows affecting less than a whole physical core to a VM (tiny VMs). The execution time values range between 1 hour to 10 hours. Those values are the common ones in terms of reservation. They are oscillating between a short and a long reservation.

Regarding the interval values of the hosts they range between a private computer (1 core, 2 GB memory, 1 Ghz clock frequency) and a cluster (24 cores, 24 GB memory, 3 Ghz clock frequency). This is done to encompass all types of machines that could compose a cloud. Note that the CPU and the memory values are related. In other words, a machine with a high number of CPUs will have high memory capacity and *vice versa*.

5.1.2. Real Hardware Settings

In the real experiments we generated real VM templates for requesting VMs, as a real user has to do when he/she fills the VM template. The VMs features (execution time, number of CPUs and memory) were generated following a poisson distribution with respectively the λ parameters (1, 2, 4). The sample of each VM feature is composed of 12000 values. The used results for each distribution are taken in the intervals [1, 10] for the execution time, [1, 9] for the CPU and [1, 8] for the memory. The total number of generated VMs is 2000.

Regarding the hosts, we used machines from Grid'5000 [29]. We used the site of Nancy equipped with clusters allowing the virtualization to deploy OpenNebula with and without EMLS-ONC. The hardware specifications of the hosts that compose our cloud are summarized in Table 1.

Table 1: Hardware Specifications of the Hosts

Device Specification	Value
Max Number of hosts	200
Frontend hosts	1
Number of clusters	2
Clock frequency	2.5 Ghz, 2.53 Ghz
Memory capacity	8192 Mo, 16384 Mo
Number of cores	8, 4
CPU type	Intel Xeon E5420, Intel Xeon X3440

5.2. Algorithm and Instances Parameters

5.2.1. Artificial Experimental Parameters

The EMLS-ONC and EMLS-ONC-MO schedulers are proposed to be integrated in the OpenNebula cloud manager. Therefore, they have to be flexible and fit different cloud configurations. In this context, we conducted our experiments on both approaches during 20 *scheduling cycles* for each instance. We define an instance as a certain number of VMs that arrive per scheduling cycle on a certain number of hosts. In these experiments, we compare the EMLS-ONC with an energy-aware algorithm, while the multi-objective EMLS-ONC-MO is compared to both an energy-aware and a VM performance-aware algorithm. Note that comparison of both approaches is also done with the default OpenNebula scheduler. We measure in these experiments their ability to handle a flow of VMs requests arriving at the same time (one scheduling cycle) by comparing one to one the results of each scheduling cycle. We also discuss the results of each algorithm over several scheduling cycles in row. The comparison study of the single objective algorithm EMLS-ONC concerns the number of assigned VMs, the energy consumption and the processing time duration criteria. Regarding the Pareto EMLS-ONC-MO scheduler, the study concerns the VM performance criterion in addition to the previously cited criteria. The constraint for both schedulers is to provide the results before the end of the scheduling cycle time, and thus, the arrival of a new pool of VMs. Moreover, all the experiments presented below deal with the scheduling process part of the algorithm and nohow with the physical VMs dispatching phase. The VM dispatching phase is handled by OpenNebula.

In our experiments, we used some parameters: the scheduling cycle duration, number of scheduling cycles in row, number of VMs per arrival in each scheduling cycle, and the number of hosts composing the cloud. We performed experiments with 4 different cloud configurations, from a small local cloud with 5 hosts to a wide distributed cloud with 320 machines. Concerning varying the VMs arrival rates we use 5 different loads from a single VM to a massive arrival of 100 VMs on each scheduling cycle. Each instance is defined to be the couple (# VMs, # hosts) during 20 cycles of 30 seconds. This represents for the highest work load (100VMs/scheduling cycle) 2000 VMs for each cloud configuration. The constant value scheduling cycle duration is deduced from OpenNebula. Indeed, this value is the default waiting time period between two scheduling phases. Note that this is fixed by default to 30 seconds but it can be changed according to the needs. The other constant value which is the number of scheduling cycles in row has been determined from experimentations. We noticed that 20 scheduling cycles in row was the number of cycles needed to saturate the biggest cloud configuration (320 hosts) with the highest VMs arrival per scheduling cycle (100 VMs). A full description of the experiments parameters is given in Table 2.

Table 2: Artificial Experimental Parameters

Parameter	Value
Scheduling cycle	30s
Number of scheduling cycles in row	20
Number of VM per arrival	1, 5, 20, 60, 100
Number of hosts	5, 20, 80, 320

5.2.2. Real Experimental Parameters

In the real experiments we used 2 types of clouds, a medium one (50 hosts) and a big one (200 hosts). The clouds have been deployed on the Grid5000 platform [29]. The VMs arrival workloads during each scheduling cycle are 5, 50 and 100. The objective of this experiment is to prove the feasibility of the deployment of our approach over physical machines, while handling their constraints (lags, communications,...). The behaviors of EMLS-ONC and its performances were compared to default OpenNebula scheduler. We treat all configurations to show the different algorithm behaviors. We expressed the couples (number of hosts, number of VMs) as ratios. A ratio is the relationship between the number of VMs to be assigned and the number of hosts that compose the cloud. Thus, we dealt with a small ratio (50 hosts, 5 VMs), a big ratio (50 hosts, 100 VMs) and a medium ratio (200 hosts, 100 VMs). The scheduling cycle duration and the number of scheduling cycles in row have been fixed for the same reason as mentioned in Section 5.2.1. We used 30 seconds as time duration for the scheduling cycle in all the instances except in the (200 hosts, 100 VMs) instance. We used 60 seconds in this latter because of the communication delays due to a large cloud (200 hosts) which increases the processing time of EMLS-ONC. The scheduling cycles in row value was fixed to 10 because that was sufficient to saturate the 200 hosts cloud. A full description of the experiments parameters is given in Table 3.

Table 3: Real Experimental Parameters

Parameter	Value
Scheduling cycle	30s, 60s
Number of scheduling cycles in row	10
Number of VM per arrival	5, 50, 100
Number of hosts	50, 200

5.3. Experimental Results

In the following, we discuss the experiment study of two scheduling approaches based on a metaheuristic, integrating both the energy consumption and the VM performances on the top of OpenNebula cloud manager. We perform a set of experiments with different parameters cited before in both Section 5.2.1 and 5.2.2 respectively for the artificial and the real experiments.

The first single objective approach EMLS-ONC has been compared to two algorithms. The first comparison was done with an energy-aware FFD based algorithm. The FFD algorithm as said previously has been proved to perform very well in energy savings. The second comparison was done obviously with the default OpenNebula scheduler to know the impact of our approach on the OpenNebula cloud manager. The default scheduler of OpenNebula is based on a type of consolidation technique. This technique provides energy consumption reduction. In addition, we

validated through experiments on a real cloud infrastructure (GRID5000) the behavior and the contributions of the EMLS-ONC over the default OpenNebula scheduler when deployed on real hosts.

The second approach, the Pareto EMLS-ONC-MO, has been compared to three algorithms. Indeed, dealing with two objectives, we compared EMLS-ONC-MO to two heuristics tackling each one an objective among the two. We compared EMLS-ONC-MO with the default OpenNebula scheduler as well. The two heuristics mentioned earlier are both based on FFD assignment. As presented in Section 4.2, the first one is energy-aware and sorts the hosts and the VMs according to this criterion, while the second one is memory-aware in order to save the VMs performance. Thus, it sorts the hosts and the VMs according to this other criterion.

Due to the stochasticity of both EMLS-ONC and EMLS-ONC-MO, we run each experiment for each instance 30 times. The presented results for each instance of both EMLS-ONC and EMLS-ONC-MO algorithms are the average value of all the 30 executions. Note that, the noted *average values* rows in the tables of results represent the average value of each column over all the instances. In addition, the notation *# VMs* on the type of instance column represents the number of VMs that arrive per scheduling cycle. Moreover, in the tables of results the *RPC* acronym designates the *Relative Percentage Change*. It represents the percentage of difference between the obtained values of our algorithm and the values of the compared with algorithm. In our case, since dealing with minimization, negative values mean the improvements of our algorithm while positive values mean worst results.

In the presented results we also used two terms: *cumulative* and *normalized*. Those terms are due to the difference in the number of assigned VMs between the compared approaches. Indeed, it is not sufficient to compare the obtained values for each criterion at each scheduling cycle. We used the cumulative sum to link the results of each scheduling cycles and to have the evolution of the addressed criterion through the different VMs arrival waves. Moreover, the normalization uses the cumulative sum of the criterion and the cumulative number of assigned VMs to give at each moment the value per assigned VM of the addressed criterion.

The results will be discussed in four sections depending on the number of objectives or the type of experiments (artificial or real). The first section deals with single objective approaches treating the energy issue, the second with the Pareto bi-objective EMLS-ONC-MO approach compared with energy-aware algorithm, the third with the same Pareto bi-objective EMLS-ONC-MO approach but compared with VMs performance-aware algorithm and finally the fourth with the feedbacks obtained from the experiments on a real infrastructure.

The following figures show the results of 3 most specific instances. One for small cloud and small number of VMs (20 hosts, 5 VMs), one for a medium cloud with big number of VMs (80 hosts, 100VMs) and the last for big cloud with big number of VMs (320 hosts, 100 VMs), to cover all scenarios.

5.3.1. Comparison study between the single objective EMLS-ONC, the energy-aware FFD approach and the OpenNebula scheduler

In this section, we discuss the results of EMLS-ONC, the energy-aware FFD approach and the OpenNebula scheduler.

- *Number of Scheduled VMs:* Table 4 shows that EMLS-ONC assigns more VMs than the energy-aware FFD approach in average. This trend is also confirmed on the instances individually. However, despite a policy that maximizes the number of assigned VMs in the EMLS-ONC, the OpenNebula scheduler still assigns slightly more VMs. This is due to the

consolidation nature of the OpenNebula scheduler which keeps more space during the successive scheduling cycles for the next VMs arrivals. Indeed, in the first scheduling cycles EMLS-ONC manages to keep up on the number of assigned VMs with the OpenNebula scheduler but it collapses at the end. This phenomenon is more significant with the high VMs arrivals when assigned over a proportionally small cloud. This behavior is caused by the previous assignments which ultimately govern the next ones.

- *Processing time during each scheduling cycle:* in this part, we are going to talk about the time computation comparison between the approaches. Table 5 shows that the processing time of both the energy-aware FFD algorithm and the default OpenNebula scheduler are significantly lower than the scheduling cycle time. This is due to the simplicity of the algorithms. We present in this same table the maximum, minimum and the average processing time values. The results show that despite the complexity of EMLS-ONC the processing time value never exceeds the default scheduling cycle duration (Max value) and it computes really fast for small instances (Min value).

- *Energy consumption savings:* in Figure 11 and Figure 12 we notice that the EMLS-ONC energy histogram is always under the OpenNebula scheduler and the energy-aware FFD algorithm ones, which means better energy reduction during the whole scheduling cycles. Nevertheless, in Figure 14 we notice that after a good start, EMLS-ONC becomes slightly less efficient than the energy-aware FFD algorithm between the scheduling cycles 7 and 12. It becomes better again until the end of the scheduling cycles. This is due to the difference in the assignment policies of the two approaches. Indeed, the EMLS-ONC assigns more VMs in that period of time (see Figure 13) which leads to an energy consumption increase. However, the additional assigned VMs through the next scheduling cycles is done efficiently which brings down the energy consumption of the EMLS-ONC. Note that, the energy consumption drawn in the figures and mentioned earlier is a normalized value. It gives the energy consumption per VM through the different scheduling cycles by taking into account both the number of assigned VMs and the total energy consumption. This metric helps to show the real energy efficiency of each approach.

  In addition, we note in all the previous figures that EMLS-ONC has an increasing evolution. It changes according to the load and clearly shows its superiority over the first few cycles when the VM assignment remains subject to choice. In contrast, the other approaches have a downward evolution mainly OpenNebula scheduler (consolidation). It has a bad energy assignment at the beginning and starts to stabilize its energy consumption only on the last cycles when the load is at its highest level. We also note that the Energy-aware approach FFD is better on average than the OpenNebula scheduler on non-dense instances and a little bit worse on the instances with high loads as shown in the last cycles of Figure 12.

  Table 4 reports none normalized values. Therefore, we note that with the two instances (320 hosts, 60 VMs) and (320 hosts, 100 VMs), EMLS-ONC has worse results than energy-aware FFD. This is not relevant because EMLS-ONC has a significantly higher number of assigned VMs. We can see then the interest of using the normalized values in the figures.

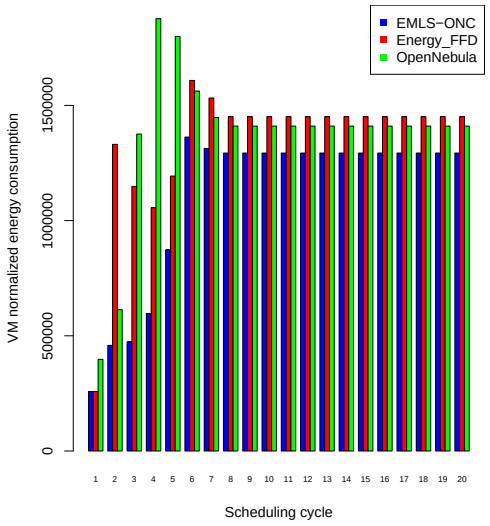

Figure 11: Comparison of the results of the normalized energy consumption during 20 scheduling cycles in row between EMLS-ONC scheduler, energy-aware FFD scheduler and OpenNebula scheduler for a configuration of 5 VMs and 20 hosts.

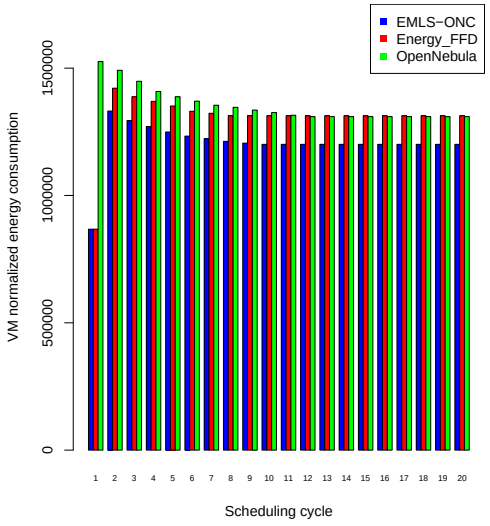

Figure 12: Comparison of the results of the normalized energy consumption during 20 scheduling cycles in row between EMLS-ONC scheduler, energy-aware FFD scheduler and OpenNebula scheduler for a configuration of 100 VMs and 80 hosts.

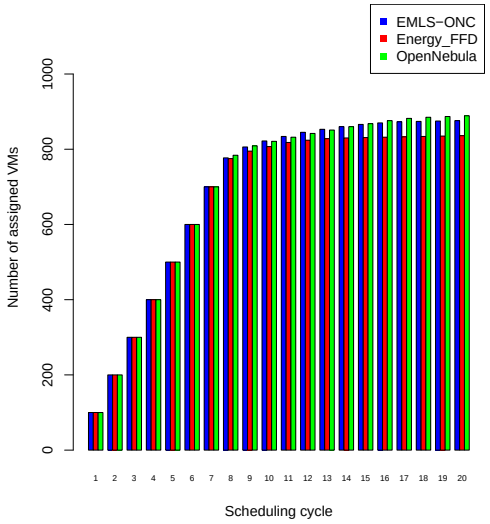

Figure 13: Comparison of the number of scheduled VMs during 20 scheduling cycles in row between EMLS-ONC scheduler, energy-aware FFD scheduler and OpenNebula scheduler for a configuration of 100 VMs and 320 hosts.

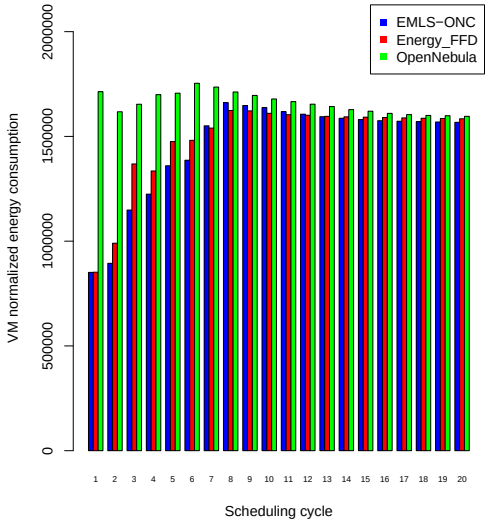

Figure 14: Comparison of the results of the normalized energy consumption during 20 scheduling cycles in row between EMLS-ONC scheduler, energy-aware FFD scheduler and OpenNebula scheduler for a configuration of 100 VMs and 320 hosts.

Table 4: Comparison between the results obtained by the EMLS-ONC, the energy-aware FFD algorithm and the OpenNebula scheduler

Type of Instance		EMLS-ONC vs Energy-FFD		EMLS-ONC vs OpenNebula	
# Hosts	# VMs	# of Additional Assigned VMs	Energy Consumption RPC	# of Additional Assigned VMs	Energy Consumption RPC
5	1	**0**	**0%**	**0**	**0%**
5	5	**0**	**-25.40%**	**-4**	**-13.98%**
5	20	**1**	**-38.67%**	**-2**	**-0.83%**
5	60	**2**	**-11.75%**	**0**	**-18.25%**
5	100	**2**	**-6.28%**	**-1**	**-24.99%**
20	1	**0**	**-69.31%**	**0**	**-61.10%**
20	5	**-1**	**-13.51%**	**-1**	**-11.00%**
20	20	**-2**	**-27.89%**	**-4**	**-26.14%**
20	60	**1**	**-16.96%**	**-8**	**-1.55%**
20	100	**5**	**-39.99%**	**-5**	**-9.84%**
80	1	**0**	**-73.50%**	**0**	**-87.17%**
80	5	**0**	**-23.57%**	**0**	**-29.25%**
80	20	**7**	**-5.97%**	**-3**	**-8.52%**
80	60	**17**	**-8.52%**	**-11**	**-9.57%**
80	100	**10**	**-4.25%**	**-8**	**-11.48%**
320	1	**0**	**-77.92%**	**0**	**-88.14%**
320	5	**0**	**-57.59%**	**0**	**-77.90%**
320	20	**0**	**-2.84%**	**0**	**-29.84%**
320	60	**13**	**1.23%**	**-17**	**-4.23%**
320	100	**40**	**3.67%**	**-13**	**-3.23%**
Average values		**4.75**	**-25%**	**-3.85**	**-26%**

Table 5: Comparison between the processing time during 20 scheduling cycle in a row of the EMLS-ONC, the energy-aware FFD and the OpenNebula scheduler

Computation time (sec)	EMLS-ONC Scheduler	Energy-FFD Scheduler	OpenNebula Scheduler
Max value	**24.5178**	**0.541404**	**0.0264464**
Min value	**0.00219981**	**0.00189716**	**0.00184849**
Average value	**1.18063915**	**0.03622298**	**0.004775**

5.3.2. Comparison study between the Pareto EMLS-ONC-MO, the energy-aware FFD approach and the OpenNebula scheduler

In this section, we discuss the results of EMLS-ONC-MO, the energy-aware FFD approach and the OpenNebula scheduler.

- *Number of Scheduled VMs:* Table 6 shows that EMLS-ONC-MO assigns more VMs than the energy-aware FFD approach in average. This trend is also confirmed on the instances individually. However, EMLS-ONC-MO is less efficient when compared to the OpenNebula scheduler but still better on average. This is due to the consolidation nature of the OpenNebula scheduler which keeps more space during the successive scheduling cycles for the next VMs arrivals. We notice also that EMLS-ONC-MO assigns significantly more VMs than the other approaches on instances where the architecture is large compared to the number of VMs (# hosts >> # VMs) like for instances (20 hosts, 5 VMs),(320 hosts, 60 VMs) and (320 hosts, 100 VMs).

- *Processing time during each scheduling cycle:* we compare here the three scheduling approaches in terms of execution time. Table 7 shows that the processing time of both the

energy-aware FFD algorithm and the default OpenNebula scheduler are significantly lower than the scheduling cycle time. This is due to the simplicity of the algorithms. We report in this table the maximum, minimum and the average processing time values. The results show that despite the complexity of EMLS-ONC-MO the processing time value never exceeds the default scheduling cycle duration (Max value) and the algorithm is fast for small instances (Min value).

- *Energy consumption and VMs performance savings:* we define the VMs performance by their response time, which is itself related to the memory usage. Therefrom, the VM response time is inversely related to the VM performance. In other words, reducing the response time of a VM means a good VM performance. In Figure 15 and Figure 19 for the energy criterion and, Figure 16 and Figure 20 for the VM performance criterion, we notice that when there are different assignment choices (i.e. during the first scheduling cycles) in the instance types (# hosts >> # VMs) such as (20 hosts, 5 VMs) and (320 hosts, 100 VMs), EMLS-ONC-MO suffers a bit from the diversity of the Pareto space. It struggles during the first scheduling cycles but ends up having better results in both VM response time and energy consumption. Moreover, it gives better results on both criteria during all the scheduling cycles on the high loaded instances such as (80 hosts, 100 VMs) (see Figure 17 and Figure 18).

  Table 6 shows that EMLS-ONC-MO is never Pareto dominated in any type of instance. We also notice that when the solutions proposed by EMLS-ONC-MO does not dominate the solutions proposed by the other approaches they still have an advantage in comparison to the other ones. Indeed, as shown in the comparison of the average values of both EMLS-ONC-MO and the energy-aware FFD (-19 % 1 %), when the EMLS-ONC-MO and the energy-aware FFD solutions do not dominate each others, the advantage that the solutions of EMLS-ONC-MO have in one objective (e.g. Energy) is more significant than the disadvantage that they may have in the other objective (e.g. VM performance).

  We also note that EMLS-ONC-MO has a tendency to improve more significantly the energy consumption than the VMs performances. This is mainly due to the fact that an energy-aware assignment can be determined from the beginning while a more focused VMs performance assignment needs to wait for the last scheduling cycles where the VMs performance start to decrease because of the load. This leads to a shorter optimization time dedicated to VM performance with less assignment possibilities.

  As for EMLS-ONC, Table 6 shows that with the two instances (320 hosts, 60 VMs) and (320 hosts, 100 VMs), EMLS-ONC-MO has worse results than energy-aware FFD in both objectives. This is not relevant because Table 6 reports none normalized values. Those values do not take into account the number of assigned VMs which is significantly higher for EMLS-ONC-MO. We can see then the interest of using the normalized values in the figures.

5.3.3. Comparison study between the Pareto EMLS-ONC-MO, the memory-aware FFD approach and the OpenNebula scheduler

In this section, we discuss the results of EMLS-ONC-MO, the memory-aware FFD approach and the OpenNebula scheduler.

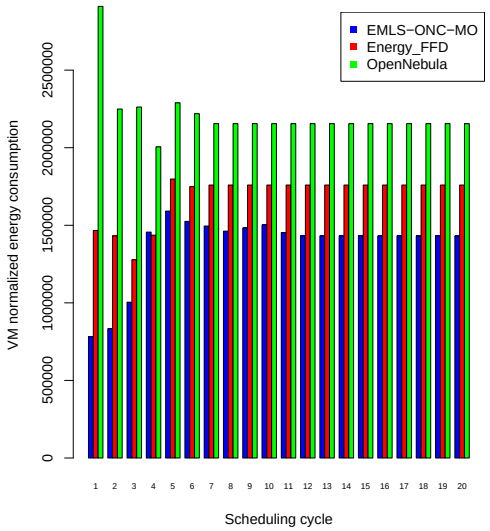

Figure 15: Comparison of the results of the normalized energy consumption during 20 scheduling cycles in row between EMLS-ONC-MO scheduler, energy-aware FFD scheduler and OpenNebula scheduler for a configuration of 5 VMs and 20 hosts.

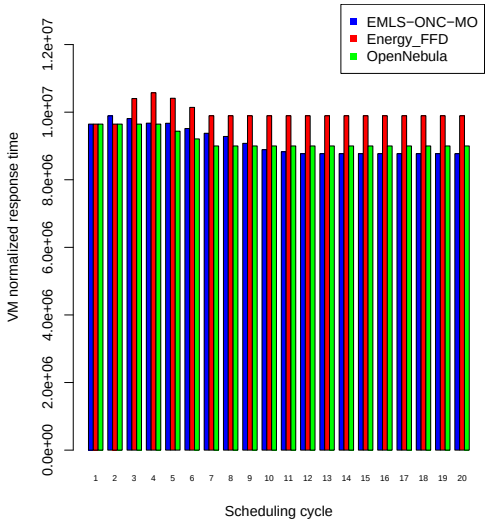

Figure 16: Comparison of the results of the normalized response time of a VM during 20 scheduling cycles in row between EMLS-ONC-MO scheduler, energy-aware FFD scheduler and OpenNebula scheduler for a configuration of 5 VMs and 20 hosts.

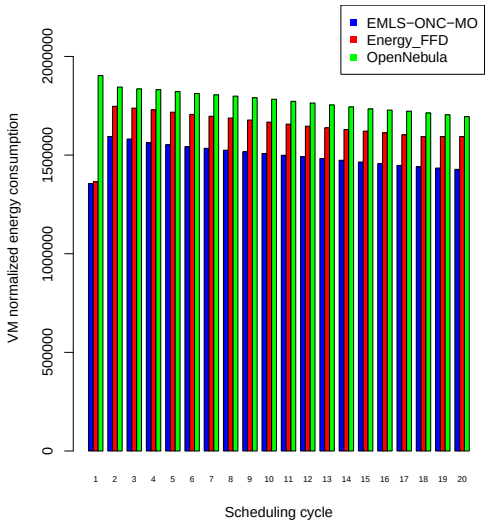

Figure 17: Comparison of the results of the normalized energy consumption during 20 scheduling cycles in row between EMLS-ONC-MO scheduler, energy-aware FFD scheduler and OpenNebula scheduler for a configuration of 100 VMs and 80 hosts.

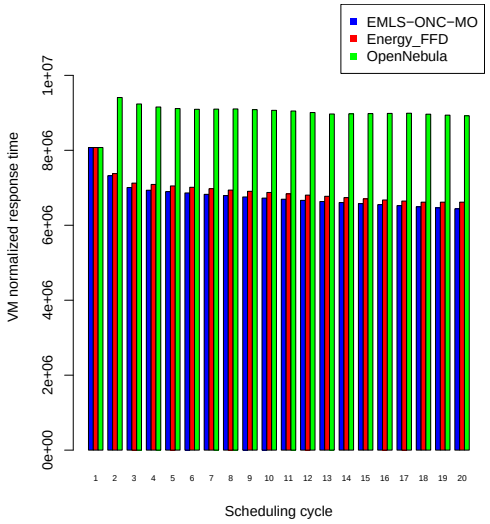

Figure 18: Comparison of the results of the normalized response time of a VM during 20 scheduling cycles in row between EMLS-ONC-MO scheduler, energy-aware FFD scheduler and OpenNebula scheduler for a configuration of 100 VMs and 80 hosts.

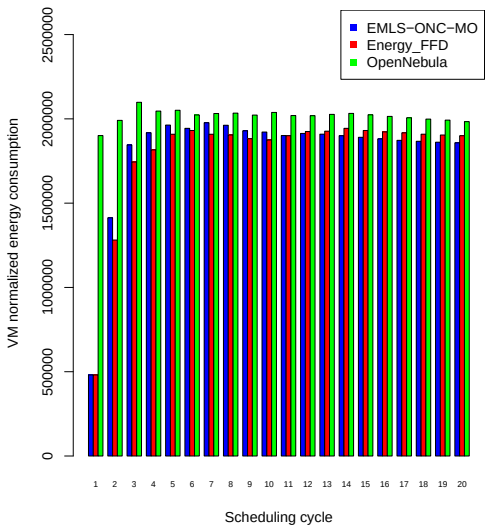

Figure 19: Comparison of the results of the normalized energy consumption during 20 scheduling cycles in row between EMLS-ONC-MO scheduler, energy-aware FFD scheduler and OpenNebula scheduler for a configuration of 100 VMs and 320 hosts.

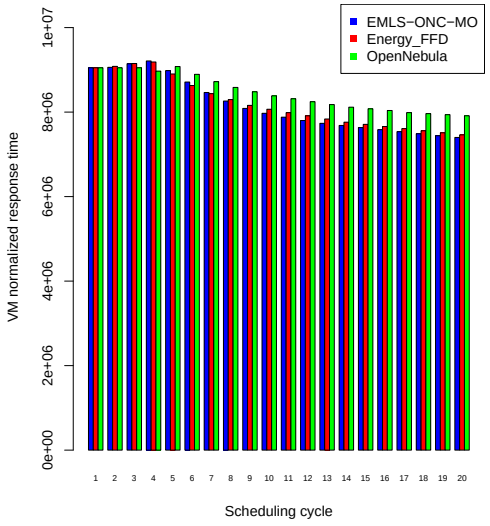

Figure 20: Comparison of the results of the normalized response time of a VM during 20 scheduling cycles in row between EMLS-ONC-MO scheduler, energy-aware FFD scheduler and OpenNebula scheduler for a configuration of 100 VMs and 320 hosts.

Table 6: Comparison between the results obtained by the EMLS-ONC-MO, the energy-aware FFD algorithm and the OpenNebula scheduler

Type of instance		EMLS-ONC-MO vs Energy-FFD			EMLS-ONC-MO vs OpenNebula		
# Hosts	# VMs	# of Additional Assigned VMs	Energy Consumption RPC	VMs Response Time RPC	# of Additional Assigned VMs	Energy Consumption RPC	VMs Response Time RPC
5	1	**0**	**0%**	**0%**	**0**	**0%**	**0%**
5	5	**0**	**-0.59%**	**-1.39%**	**0**	**-0.78%**	**-19.98%**
5	20	**2**	**-9.61%**	**-1.03%**	**1**	**-6.06%**	**-9.45%**
5	60	**2.25**	**-55.90%**	**1.50%**	**-3.75**	**-39.51%**	**0.28%**
5	100	**1**	**-56.22%**	**1.75%**	**-4**	**-43.29%**	**-2.41%**
20	1	**0**	**0%**	**0%**	**0**	**0%**	**0%**
20	5	**2**	**-6.59%**	**5.80%**	**3**	**-14.81%**	**-19.62%**
20	20	**2**	**-11.61%**	**0.54%**	**1**	**-18.79%**	**10.69%**
20	60	**-1**	**-40.06%**	**1.19%**	**-9**	**-19.11%**	**-7.00%**
20	100	**3**	**-5.42%**	**-0.83%**	**-6**	**-10.94%**	**-10.97%**
80	1	**0**	**-62.30%**	**-6.51%**	**0**	**-82.50%**	**0%**
80	5	**0**	**-6.64%**	**0.51%**	**0**	**-29.02%**	**-27.86%**
80	20	**2**	**-10.59%**	**0.90%**	**-1**	**-17.99%**	**-31.67%**
80	60	**10**	**-0.57%**	**4.57%**	**-7**	**-13.24%**	**-17.48%**
80	100	**5**	**-7.69%**	**0.26%**	**-5**	**-18.17%**	**-29.90%**
320	1	**0**	**-71.41%**	**0%**	**0**	**-86.65%**	**0%**
320	5	**0**	**-19.78%**	**0.45%**	**0**	**-55.59%**	**2.44%**
320	20	**0**	**-32.35%**	**3.73%**	**0**	**-17.96%**	**8.36%**
320	60	**35.75**	**4.45%**	**3.17%**	**36.75**	**-2.62%**	**-17.32%**
320	100	**34**	**3.44%**	**4.91%**	**19**	**-3.39%**	**-3.54%**
Average values		**4.9**	**-19%**	**1%**	**1.25**	**-24%**	**-9%**

Table 7: Comparison between the processing time during 20 scheduling cycle in a row of the EMLS-ONC-MO, the energy-aware FFD and the OpenNebula scheduler

Computation Time (sec)	EMLS-ONC-MO Scheduler	Energy-FFD Scheduler	OpenNebula Scheduler
Max value	**15.430325**	**0.57012**	**0.103329**
Min value	**0.00221455**	**0.00189864**	**0.00181377**
Average value	**0.73919431**	**0.04525467**	**0.00500326**

- *Number of Scheduled VMs:* Table 8 shows that EMLS-ONC-MO assigns more VMs than the memory-aware FFD approach in average. This trend is also confirmed on the instances individually. We note also that the improvement of EMLS-ONC-MO in the VM assignments are more significant against memory-aware FFD than against energy-aware FFD. However, EMLS-ONC-MO gives in average almost the same results as OpenNebula scheduler. This is due as said previously to the consolidation nature of the OpenNebula scheduler which keeps more space during the successive scheduling cycles for the next VMs arrivals. In addition, we notice that EMLS-ONC-MO outperforms memory-aware FFD on the instances where the infrastructure is relatively large (> 20 hosts) especially for high load (> 60VMs per cycle) like in instances (20 hosts, 100 VMs),(80 hosts, 60 VMs) and (320 hosts, 100 VMs). We deduce that a memory assignment policy misuses the available space of the resources.

- *Processing time during each scheduling cycle:* we compare here the three scheduling approaches in terms of execution time. Table 9 shows that the processing time of both the memory-aware FFD algorithm and the default OpenNebula scheduler is significantly lower

than the scheduling cycle time. This is due as said before to the low complexity of the algorithms. We presented in this table the maximum, minimum and the average processing time values. The results show that despite the complexity of EMLS-ONC-MO the processing time value never exceeds the default scheduling cycle duration (Max value)and the algorithm is fast for small instances (Min value).

- *Energy consumption and VMs performance savings:* as previously said, the VMs performance is related to the memory usage. Therefore, memory-aware FFD addresses the VMs performance issue.

  We note that memory-aware FFD is very efficient in the performance of VMs (Figure 22 and Figure 26). However, this efficiency has a drawback over the other criterion (i.e. the energy consumption) (see Figure 21 and Figure 25). Memory-aware FFD reaches this high efficiency in the VMs performance on the instances where the cloud size is proportionally greater than the number of arriving VMs. This is due to the possibility in that type of instances of assigning VMs without overloading the hosts. Always for the same type of instances we note that EMLS-ONC-MO performs very well for energy reduction and is quite close to memory-aware FFD on the VMs performance criterion. Nevertheless, we observe that the more the energy reduction is significant (see Figure 21) the less efficient EMLS-ONC-MO is, regarding the VMs performances (see Figure 22).

  Moreover, we notice that for instances with high VMs load such as in Figure 23, the first assignment (1st scheduling cycle) of EMLS-ONC-MO gives the best result compared to the other approaches, but it leads to less energy efficient results during the next scheduling cycles. However, EMLS-ONC-MO finds the tradeoff and makes up with the improvement of the VM performances until being better than the memory-aware FFD (see Figure 24). We can see here the advantage of a Pareto EMLS-ONC-MO approach.

  Moreover, Table 8 shows that EMLS-ONC-MO is never Pareto dominated for any type of instance and is in average better than both memory-aware FFD (-14%, 2%) and OpenNebula scheduler (-25%, -10%). The only times where the solutions proposed by EMLS-ONC-MO are dominated by the solutions of one of the other approaches, are when EMLS-ONC-MO assigns more VMs than the other approaches. The normalized values drawn in the figures prove the Pareto superiority of EMLS-ONC-MO over memory-aware FFD and OpenNebula scheduler when including the number of assigned VMs. We note also that the advantage of EMLS-ONC-MO over memory-aware FFD is less significant than the one obtained over energy-aware FFD. Indeed, memory-aware FFD is more comprehensive. It performs well on the VM performances criterion, in addition to improving sometimes the energy consumption. This is due to the relationship between the memory and the CPU capacity. In other words, a host with a high memory features has also good CPU features. Therefore, sorting the hosts by memory is equivalent to sorting them from the one with the best CPU capacity to the one with the worst. We deduce then that VM assignment using this technique, consolidates the VMs in the hosts from the biggest to the smallest one. This consolidation is the reason why memory-aware FFD has also good energy results.

5.3.4. Comparison study between the EMLS-ONC and the OpenNebula scheduler through a real deployment on GRID5000 infrastructure

In this section, we discuss the comparison between the results of both algorithms EMLS-ONC and the OpenNebula default scheduler over GRID5000. The results are presented in Ta-

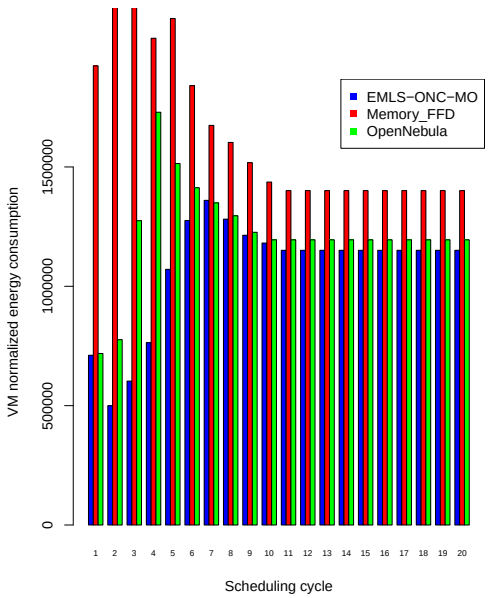

Figure 21: Comparison of the results of the normalized energy consumption during 20 scheduling cycles in row between EMLS-ONC-MO scheduler, memory-aware FFD scheduler and OpenNebula scheduler for a configuration of 5 VMs and 20 hosts.

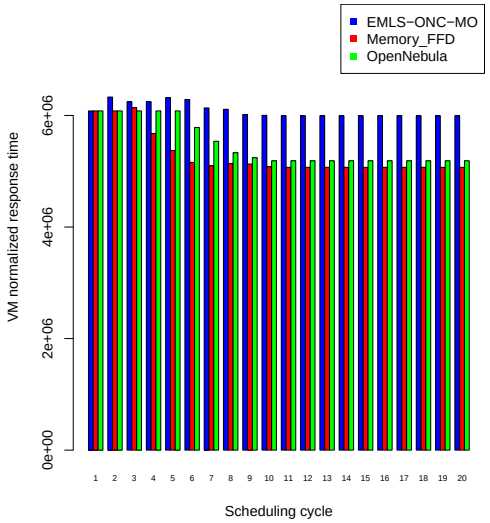

Figure 22: Comparison of the results of the normalized response time of a VM during 20 scheduling cycles in row between EMLS-ONC-MO scheduler, memory-aware FFD scheduler and OpenNebula scheduler for a configuration of 5 VMs and 20 hosts.

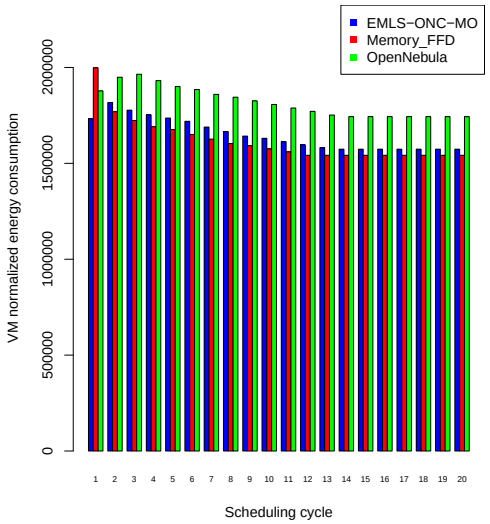

Figure 23: Comparison of the results of the normalized energy consumption during 20 scheduling cycles in row between EMLS-ONC-MO scheduler, memory-aware FFD scheduler and OpenNebula scheduler for a configuration of 100 VMs and 80 hosts.

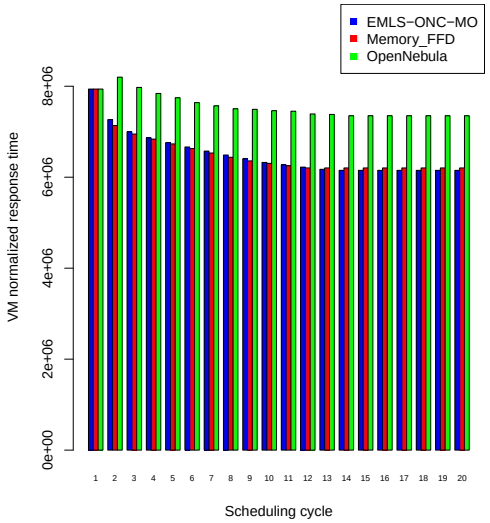

Figure 24: Comparison of the results of the normalized response time of a VM during 20 scheduling cycles in row between EMLS-ONC-MO scheduler, memory-aware FFD scheduler and OpenNebula scheduler for a configuration of 100 VMs and 80 hosts.

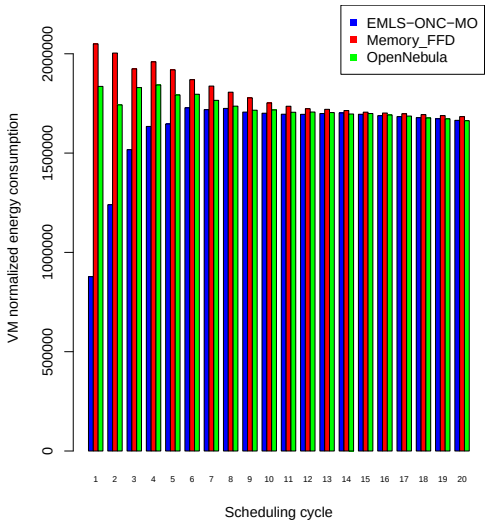

Figure 25: Comparison of the results of the normalized energy consumption during 20 scheduling cycles in row between EMLS-ONC-MO scheduler, memory-aware FFD scheduler and OpenNebula scheduler for a configuration of 100 VMs and 320 hosts.

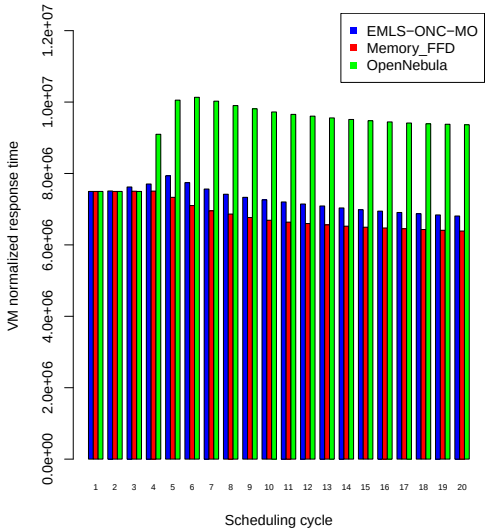

Figure 26: Comparison of the results of the normalized response time of a VM during 20 scheduling cycles in row between EMLS-ONC-MO scheduler, memory-aware FFD scheduler and OpenNebula scheduler for a configuration of 100 VMs and 320 hosts.

Table 8: Comparison between the results obtained by the EMLS-ONC-MO, the memory-aware FFD algorithm and the OpenNebula scheduler

Type of instance		EMLS-ONC-MO vs Memory-FFD			EMLS-ONC-MO vs OpenNebula		
# Hosts	# VMs	# of Additional Assigned VMs	Energy Consumption RPC	VMs Response Time RPC	# of Additional Assigned VMs	Energy Consumption RPC	VMs Response Time RPC
5	1	**0**	**0%**	**0%**	**0**	**0%**	**0%**
5	5	**0**	**-22.64%**	**0%**	**0**	**-10.62%**	**0%**
5	20	**0**	**17.28%**	**-30.77%**	**-1**	**2.34%**	**-1.82%**
5	60	**0**	**-26.99%**	**0%**	**-5**	**-65.22%**	**3.18%**
5	100	**3**	**-30.73%**	**0%**	**-4**	**-35.32%**	**-0.61%**
20	1	**0**	**-48.92%**	**-0.89%**	**0**	**-63.77%**	**-47.08%**
20	5	**0**	**-17.84%**	**18.24%**	**0**	**-3.66%**	**15.56%**
20	20	**1**	**-5.90%**	**2.68%**	**-1**	**-15.49%**	**9.14%**
20	60	**11**	**3.22%**	**-0.32%**	**-1**	**-16.69%**	**-5.84%**
20	100	**22**	**47.37%**	**-2.51%**	**-4**	**0.90%**	**-13.53%**
80	1	**0**	**-64.77%**	**0%**	**0**	**-79.65%**	**0%**
80	5	**5**	**-9.69%**	**14.62%**	**-2**	**-14.56%**	**-7.44%**
80	20	**3**	**-7.09%**	**7.97%**	**-3**	**-29.79%**	**-36.98%**
80	60	**23**	**-0.56%**	**4.15%**	**-5**	**-8.65%**	**-22.43%**
80	100	**4.5**	**4.71%**	**1.71%**	**-2.5**	**-10.98%**	**-17.48%**
320	1	**0**	**-84.69%**	**0%**	**0**	**-90.43%**	**0%**
320	5	**0**	**-38.12%**	**0.68%**	**0**	**-38.48%**	**0.68%**
320	20	**5**	**-1.37%**	**11.78%**	**-6**	**-9.24%**	**-25.22%**
320	60	**12**	**0.89%**	**9.77%**	**17**	**-9.85%**	**-21.67%**
320	100	**17.5**	**1.50%**	**9.39%**	**8.5**	**1.38%**	**-26.37%**
Average values		**5.35**	**-14%**	**2%**	**-0.45**	**-25%**	**-10%**

Table 9: Comparison between the processing time during 20 scheduling cycle in a row of the EMLS-ONC-MO, the memory-aware FFD and the OpenNebula scheduler

Computation Time (sec)	EMLS-ONC-MO Scheduler	Memory-FFD Scheduler	OpenNebula Scheduler
Max value	**16.5711**	**0.530779**	**0.0211479**
Min value	**0.00224968**	**0.00188564**	**0.00178865**
Average value	**0.70908986**	**0.04163962**	**0.00480393**

ble 10 to Table 15. For each instance we present two tables, one for the raw results and the other for the normalized results.

- *Number of Scheduled VMs:* the results show that when EMLS-ONC is deployed on a real infrastructure, it assigns more VMs than the default scheduler of OpenNebula. We do not observe a difference in the number of assigned VMs for the small ratio instances (20 hosts, 5 VMs) because of the small number of arriving VMs compared to the size of the cloud. All the VMs could be assigned in both approaches (see Table 10). However, for medium and high ratio instances, EMLS-ONC is better in the number of assigned VMs as presented in Table 12 and Table 14.

- *Computation time during each scheduling cycle:* the results show that EMLS-ONC never exceeds the scheduling cycle limit in any of the instances. For the instances where the scheduling time was set to 30 sec (i.e. (50 hosts, 5 VMs) and (50 hosts, 100 VMs)), the longest processing time registered was of 24.25*sec* (see Table 12). Regarding the last instance, the scheduling cycle time was set to 60 seconds because of the big number of VMs combined to the size of the cloud (200 hosts, 100 VMs). This increases the time

duration of the hosts filtering step in the EMLS-ONC algorithm. The obtained results show that EMLS-ONC spends for the longest processing time 53.68*sec* (see Table 14).

- *Energy consumption savings:* we can notice three different behaviors from the energy analysis of the experiments according to the type of the instance. The first analysis is the one regarding the small ratio instance (20 hosts, 5 VMs). This instance contains many assignment possibilities, we observe that EMLS-ONC improves the obtained results at each scheduling cycle with an average value of -7.81% compared to the default OpenNebula scheduler (see Table 10 and Table 11). The second analysis concerns the experiments on a big ratio instance (50 hosts, 100 VMs). We note for this type of instance an improvement during the first scheduling cycles which gradually leave room to a decrease in the improvement. The accumulation of the VMs as the scheduling cycles occur, limits the opportunities for improving the assignment (see Table 12). However, because of a higher number of assigned VMs, EMLS-ONC has better normalized energy values than the OpenNebula default scheduler (see Table 13). The third and last analysis is the one of the medium ratio instance (200 hosts, 100 VMs). We observe for this type of instance that there is an improvement during the first scheduling cycles as for the previously analyzed instances. However, the improvement decreases gradually until reaching a point where EMLS-ONC gives worse results than the OpenNebula scheduler (see Table 14 and Table 15). Even if EMLS-ONC assigns more VMs than the OpenNebula scheduler, but this is not the reason to the performance decreasing. In the following, we will explain the reason of this performance drop on the instances with a medium and high ratio. As shown in Figure 27, the policy of EMLS-ONC is to assign from the beginning the VMs with an energy-efficient way. Conversely, the default scheduler of OpenNebula fills each host to its maximum capacity and relies on the time to benefit from its assignment. One can have at a given moment as shown in Figure 27, two different assignments. The first at the right of the figure (EMLS-ONC) which minimizes at each moment the energy consumption, and the second at the left which consolidates (OpenNebula). The problem comes from the energy evaluation function in Equation (3). Indeed, to be as precise as possible, our energy function evolves according to the previous CPU usage made in the host. Therefore, Equation (3) does not give the same results for a same VM j assigned to the host i if this host i is free or if it is used at 60%. It is obvious then to see that, while the OpenNebula default scheduler and other consolidation approaches will start by over loading the first host before switching to another one, EMLS-ONC assigns with the aim not to reach a critical usage of the hosts by having a balanced assignment. With these two policies one can see that during the last scheduling cycles, the EMLS-ONC has no alternative to find free hosts. It becomes mandatory for it to assign the VMs with a high CPU_usage_i value having an exponential increase while calculating the energy consumption. On the other hand, the OpenNebula scheduler keeps some hosts free and benefits from this situation to avoid the additional energy cost given by CPU_usage_i. This does not show a realistic energy consumption for the OpenNebula scheduler. Indeed, the hosts usage features are updated only between the scheduling cycles. One can see that because of a progressive filling, EMLS-ONC is affected by the CPU_usage_i parameter, while the default OpenNebula scheduler is not because of the big number of VMs brought by each scheduling cycle that fully fill the host at once. To avoid this phenomenon, a solution would be to update the hosts parameters after each VM assignment and not after the whole scheduling cycle. That way, one can be able to see the evolution of the energy consumption fairly regardless the

policy.

Another reason for this performance drop of EMLS-ONC is the weak heterogeneity of the infrastructure that we used for the real experiments. It is composed of just two different types of machines. In contrast, this problem is less noticeable in the artificial experiments where the heterogeneity is significant. This is due to the high advantage that EMLS-ONC takes from this heterogeneity during the first scheduling cycles. Therefore, the previously cited phenomenon caused by the high loaded instances (e.g. 100 VMs per scheduling cycle) is compensated by the energetic gains from the beginning.

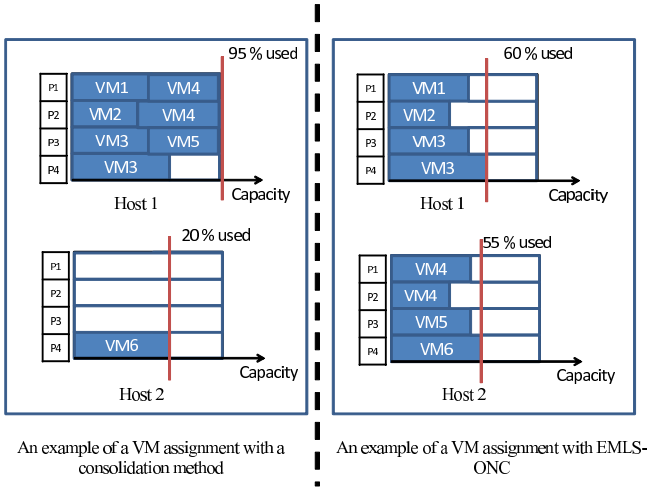

Figure 27: The difference between the assignment policies of EMLS-ONC and a consolidation based algorithm (OpenNebula scheduler) on a low heterogeneity architecture.

Table 10: Comparison between the results obtained by the EMLS-ONC and the OpenNebula scheduler for 5 VMs arrival per scheduling cycle on 50 machines of GRID5000

Type of instance		EMLS-ONC			OpenNebula			EMLS-ONC vs OpenNebula
# Hosts	# VMs	# of Assigned VMs per Cycle	Energy Consumption per Cycle	Time Processing per Cycle	# of Assigned VMs per Cycle	Energy Consumption per Cycle	Time Processing per Cycle	Energy Consumption RPC per Cycle
50	5	**5**	**618750**	**0.247646**	**5**	**644340**	**0.115171**	**-3.97%**
	5	**5**	**506250**	**0.206066**	**5**	**566238**	**0.112818**	**-10.59%**
	5	**5**	**618750**	**0.222904**	**5**	**673628**	**0.142955**	**-8.15%**
	5	**5**	**450000**	**0.235296**	**5**	**478374**	**0.10662**	**-5.93%**
	5	**5**	**562500**	**0.231712**	**5**	**615052**	**0.154184**	**-8.54%**
	5	**5**	**902326**	**0.208596**	**5**	**990916**	**0.17752**	**-8.94%**
	5	**5**	**675000**	**0.218978**	**5**	**766374**	**0.158824**	**-11.92%**
	5	**5**	**677326**	**0.242968**	**5**	**746848**	**0.119851**	**-9.31%**
	5	**5**	**675000**	**0.249436**	**5**	**722899**	**0.125968**	**-6.63%**
	5	**5**	**630382**	**0.190761**	**5**	**646875**	**0.150096**	**-2.55%**
Sum/Average	50	**50**	**631628.4**	**0.2254363**	**50**	**685154.4**	**0.1364007**	**-7.81%**

Table 11: Comparison between the cumulative results obtained by the EMLS-ONC and the OpenNebula scheduler for 5 VMs arrival on 50 machines of GRID5000

Type of instance		Cumulative Energy Consumption		Cumulative # VMs Assigned		Normalized Energy Consumption		EMLS-ONC vs OpenNebula
								Normalized Energy
# Hosts	# VMs	EMLS-ONC	OpenNebula	EMLS-ONC	OpenNebula	EMLS-ONC	OpenNebula	Consumption RPC
50	5	**618750**	**644340**	**5**	**5**	**123750**	**128868**	**-3.97%**
	5	**1125000**	**1210578**	**10**	**10**	**112500**	**121057.8**	**-7.07%**
	5	**1743750**	**1884206**	**15**	**15**	**116250**	**125613.733**	**-7.45%**
	5	**2193750**	**2362580**	**20**	**20**	**109687.5**	**118129**	**-7.15%**
	5	**2756250**	**2977632**	**25**	**25**	**110250**	**119105.28**	**-7.43%**
	5	**3658576**	**3968548**	**30**	**30**	**121952.533**	**132284.933**	**-7.81%**
	5	**4333576**	**4734922**	**35**	**35**	**123816.457**	**135283.486**	**-8.48%**
	5	**5010902**	**5481770**	**40**	**40**	**125272.55**	**137044.25**	**-8.59%**
	5	**5685902**	**6204669**	**45**	**45**	**126353.378**	**137881.533**	**-8.36%**
	5	**6316284**	**6851544**	**50**	**50**	**126325.68**	**137030.88**	**-7.81%**

Table 12: Comparison between the results obtained by the EMLS-ONC and the OpenNebula scheduler for 100 VMs arrival per scheduling cycle on 50 machines of GRID5000

Type of instance		EMLS-ONC			OpenNebula			EMLS-ONC vs OpenNebula
		# of Assigned VMs	Energy Consumption	Time Processing	# of Assigned VMs	Energy Consumption	Time Processing	Energy Consumption RPC
# Hosts	# VMs	per Cycle	per Cycle	per Cycle	per Cycle	per Cycle	per Cycle	per Cycle
50	100	**100**	**1.32E+07**	**24.2551**	**100**	**1.35E+07**	**7.669**	**-2.57%**
	100	**96**	**1.21E+07**	**15.5146**	**96**	**1.17E+07**	**4.77494**	**3.93%**
	100	**1**	**135938**	**0.0149516**	**0**	**0**	**2.20E-05**	/
	100	**0**	**0**	**0.00013059**	**0**	**0**	**6.42E-05**	/
	100	**0**	**0**	**0.00012998**	**0**	**0**	**8.33E-05**	/
	100	**5**	**437630**	**0.203641**	**0**	**0**	**8.35E-05**	/
	100	**0**	**0**	**0.0003989**	**0**	**0**	**0.00015213**	/
	100	**0**	**0**	**0.0004072**	**0**	**0**	**0.00015589**	/
	100	**0**	**0**	**0.249436**	**0**	**0**	**0.00022373**	/
	100	**0**	**0**	**0.190761**	**0**	**0**	**0.00021441**	/
Sum/Average	1000	**202**	**25871468**	**4.04295563**	**196**	**25185100**	**1.24449391**	**2.73%**

6. Conclusion

In this paper, we presented two new schedulers for the cloud manager OpenNebula using a multi-start local search algorithm. A single objective one to minimize the energy consumption called EMLS-ONC and a Pareto multi-objective one named EMLS-ONC-MO that deals with both the energy consumption of the hosts and the VMs performance. Both algorithms aim at assigning the maximum number of VMs. The energy saving of our approach exploits the heterogeneity of the hosts that compose the cloud. The performance of the VMs is related to the memory sharing issues in the virtualization process by avoiding the cache misses.

Our new approaches have been evaluated in both artificial and real clouds. EMLS-ONC as an embedded part of the OpenNebula cloud manager has been experimented on the GRID5000 infrastructure. The experiments stretch over 20 instances with 5 types of VMs load per scheduling cycle and 4 types of cloud configurations, in addition to 3 real clouds deployed over GRID5000. Each instance takes 10 to 20 scheduling cycles in row. All the EMLS-ONC and EMLS-ONC-MO experimentations are performed 30 times for each instance. The results show that EMLS-ONC improves by up to **26%** on average the results obtained by the OpenNebula's default scheduler and by up to **25%** the energy-aware FFD based approach. Moreover, EMLS-ONC-MO provides

Table 13: Comparison between the cumulative results obtained by the EMLS-ONC and the OpenNebula scheduler for 100 VMs arrival on 50 machines of GRID5000

Type of instance		Cumulative Energy Consumption		Cumulative # VMs Assigned		Normalized Energy Consumption		EMLS-ONC vs OpenNebula
								Normalized Energy
# Hosts	# VMs	EMLS-ONC	OpenNebula	EMLS-ONC	OpenNebula	EMLS-ONC	OpenNebula	Consumption RPC
50	100	**13162500**	**13509100**	**100**	**100**	**131625**	**135091**	**-2.57%**
	100	**25297900**	**25185100**	**196**	**196**	**129070.918**	**128495.408**	**0.45%**
	100	**25433838**	**25185100**	**197**	**196**	**129105.777**	**128495.408**	**0.48%**
	100	**25433838**	**25185100**	**197**	**196**	**129105.777**	**128495.408**	**0.48%**
	100	**25433838**	**25185100**	**197**	**196**	**129105.777**	**128495.408**	**0.48%**
	100	**25871468**	**25185100**	**202**	**196**	**128076.574**	**128495.408**	**-0.33%**
	100	**25871468**	**25185100**	**202**	**196**	**128076.574**	**128495.408**	**-0.33%**
	100	**25871468**	**25185100**	**202**	**196**	**128076.574**	**128495.408**	**-0.33%**
	100	**25871468**	**25185100**	**202**	**196**	**128076.574**	**128495.408**	**-0.33%**
	100	**25871468**	**25185100**	**202**	**196**	**128076.574**	**128495.408**	**-0.33%**

Table 14: Comparison between the results obtained by the EMLS-ONC and the OpenNebula scheduler for 100 VMs arrival per scheduling cycle on 200 machines of GRID5000

Type of instance		EMLS-ONC			OpenNebula			EMLS-ONC vs OpenNebula
		# of Assigned VMs	Energy Consumption	Time Processing	# of Assigned VMs	Energy Consumption	Time Processing	Energy Consumption RPC
# Hosts	# VMs	per Cycle	per Cycle	per Cycle	per Cycle	per Cycle	per Cycle	per Cycle
200	100	**100**	**1.32E+07**	**51.97**	**100**	**1.37E+07**	**7.65**	**-3.65%**
	100	**100**	**1.22E+07**	**53.68**	**100**	**1.25E+07**	**5.35**	**-2.40%**
	100	**100**	**1.54E+07**	**47.97**	**100**	**1.53E+07**	**5.58**	**0.53%**
	100	**100**	**1.47E+07**	**28.03**	**100**	**1.45E+07**	**4.71**	**1.38%**
	100	**100**	**1.38E+07**	**17.65**	**100**	**1.35E+07**	**5.30**	**2.40%**
	100	**100**	**1.48E+07**	**28.30**	**100**	**1.42E+07**	**5.77**	**3.88%**
	100	**100**	**1.32E+07**	**20.57**	**100**	**1.21E+07**	**5.72**	**9.00%**
	100	**37**	**5.93E+06**	**5.15**	**36**	**4.96E+06**	**1.91**	**19.56%**
	100	**0**	**0**	**0.24**	**0**	**0**	**0.00022**	/
	100	**0**	**0**	**0.19**	**0**	**0**	**0.00021**	/
Sum/Average	1000	**737**	**103253980**	**25.3803807**	**736**	**100836570**	**4.20241951**	**2.40%**

on average Pareto results that are **24%** and **9%** better than the default OpenNebula scheduler for respectively both the energy and the VMs performance criteria. The EMLS-ONC-MO has also been compared to two other approaches, an energy-aware one and a VMs performance one. In both cases EMLS-ONC-MO showed in average better Pareto results.

In addition, our approaches provide those results by scheduling on average the same number of VMs than the OpenNebula's previous scheduler.

Besides, the major perspectives of this work are to minimize with more impact the energy consumption by using a better energy model, including other energy consumption resources like memory and hard drives. Introducing a real time hosts' features updating in our approach could be also interesting to have a more precise energy estimate. Moreover, we plan to investigate the dynamic EMLS-ONC/EMLS-ONC-MO scheduler which will use the live migration process of OpenNebula to reassign the VMs during their running phase on different hosts to optimize more the criteria according to the new VMs arrivals. However, this will depend on the flexibility, the data transfer cost and the CPU time complexity of the VMs.

[1] J. G. Koomey, Estimating total power consumption by servers in the U.S. and the world, 2008.
[2] J. Hamilton, Cooperative expendable micro-slice servers (CEMS): Low cost, low power servers for internet-scale

Table 15: Comparison between the cumulative results obtained by the EMLS-ONC and the OpenNebula scheduler for 100 VMs arrival on 200 machines of GRID5000

Type of instance		Cumulative Energy Consumption		Cumulative # VMs Assigned		Normalized Energy Consumption		EMLS-ONC vs OpenNebula Normalized Energy Consumption RPC
# Hosts	# VMs	EMLS-ONC	OpenNebula	EMLS-ONC	OpenNebula	EMLS-ONC	OpenNebula	
200	100	**13200000**	**13700000**	**100**	**100**	**132000**	**137000**	**-3.65%**
	100	**25400000**	**26200000**	**200**	**200**	**127000**	**131000**	**-3.05%**
	100	**40800000**	**41519380**	**300**	**300**	**136000**	**138397.933**	**-1.73%**
	100	**55500000**	**56019380**	**400**	**400**	**138750**	**140048.45**	**-0.93%**
	100	**69323980**	**69519380**	**500**	**500**	**138647.96**	**139038.76**	**-0.28%**
	100	**84123980**	**83766010**	**600**	**600**	**140206.633**	**139610.017**	**0.43%**
	100	**97323980**	**95876570**	**700**	**700**	**139034.257**	**136966.529**	**1.51%**
	100	**103253980**	**100836570**	**737**	**736**	**140100.38**	**137006.209**	**2.26%**
	100	**103253980**	**100836570**	**737**	**736**	**140100.38**	**137006.209**	**2.26%**
	100	**103253980**	**100836570**	**737**	**736**	**140100.38**	**137006.209**	**2.26%**

services, in: Proceedings of 4th Biennial Conference on Innovative Date Systems Research (CIDR), Asilomar, California, USA, January.

[3] N. B. Rizvandi, J. Taheri, A. Y. Zomaya, Y. C. Lee, Linear combinations of dvfs-enabled processor frequencies to modify the energy-aware scheduling algorithms, Cluster Computing and the Grid, IEEE International Symposium on 0 (2010) 388–397.

[4] Y. Lee, A. Zomaya, Energy efficient utilization of resources in cloud computing systems, The Journal of Supercomputing (2010) 1–13. 10.1007/s11227-010-0421-3.

[5] E.-G. Talbi, Metaheuristics : from design to implementation, The Sciences Po series in international relations and political economy, John Wiley & Sons, 2009.

[6] Y. C. Lee, A. Y. Zomaya, Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling, in: CCGRID'09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 92–99.

[7] S. Srikantaiah, A. Kansal, F. Zhao, Energy aware consolidation for cloud computing, in: Proceedings of HotPower '08 Workshop on Power Aware Computing and Systems, USENIX, 2008.

[8] G. Tesauro, R. Das, H. Chan, J. O. Kephart, D. Levine, F. L. R. III, C. Lefurgy, Managing power consumption and performance of computing systems using reinforcement learning, in: NIPS.

[9] A.-C. Orgerie, L. Lefevre, J.-P. Gelas, Save watts in your grid: Green strategies for energy-aware framework in large scale distributed systems, in: Parallel and Distributed Systems, 2008. ICPADS '08. 14th IEEE International Conference on, pp. 171 –178.

[10] Y. Kessaci, N. Melab, E.-G. Talbi, A pareto-based metaheuristic for scheduling hpc applications on a geographically distributed cloud federation, Cluster Computing (2012) 1–18. 10.1007/s10586-012-0210-2.

[11] A. Verma, P. Ahuja, A. Neogi, pmapper: Power and migration cost aware application placement in virtualized systems, in: V. Issarny, R. Schantz (Eds.), Middleware 2008, volume 5346 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2008, pp. 243–264.

[12] K. Mills, J. Filliben, C. Dabrowski, Comparing vm-placement algorithms for on-demand clouds, in: Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science, CLOUDCOM '11, IEEE Computer Society, Washington, DC, USA, 2011, pp. 91–98.

[13] Openstack: open source cloud computing software, http://www.openstack.org/, 2012.

[14] D. Milojic andic and, I. M. Llorente, R. S. Montero, Opennebula: A cloud management tool, Internet Computing, IEEE 15 (2011) 11 –14.

[15] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, D. Zagorodnov, The eucalyptus open-source cloud-computing system, in: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID '09, IEEE Computer Society, Washington, DC, USA, 2009, pp. 124–131.

[16] K. Keahey, T. Freeman, J. Lauret, D. Olson, Virtual workspaces for scientific applications, J. Phys.: Conf. Ser. 78 (2007) 012038+.

[17] J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, I. M. Llorente, Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers, Future Generation Computer Systems 28 (2012) 358 – 367.

[18] J. L. Lucas-Simarro, R. Moreno-Vozmediano, R. S. Montero, I. M. Llorente, Scheduling strategies for optimal service deployment across multiple clouds, Future Generation Computer Systems (2012) –.

[19] E. Feller, L. Rilling, C. Morin, Snooze: A scalable and autonomic virtual machine management framework for private clouds, in: Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), CCGRID '12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 482–489.
[20] G. von Laszewski, L. Wang, A. J. Younge, X. He, Power-aware scheduling of virtual machines in dvfs-enabled clusters, IEEE Computer Society, New Orleans, LA, 2009, pp. 1–10.
[21] T. Burd, R. Brodersen, Energy efficient cmos microprocessor design, in: System Sciences, 1995. Proceedings of the Twenty-Eighth Hawaii International Conference on, volume 1, pp. 288–297 vol.1.
[22] P. Pillai, K. G. Shin, Real-time dynamic voltage scaling for low-power embedded operating systems, in: Proceedings of the eighteenth ACM symposium on Operating systems principles, SOSP '01, ACM, New York, NY, USA, 2001, pp. 89–102.
[23] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, N. Gautam, Managing server energy and operational costs in hosting centers, SIGMETRICS Perform. Eval. Rev. 33 (2005) 303–314.
[24] L. Wang, Y. Lu, Efficient power management of heterogeneous soft real-time clusters, in: Real-Time Systems Symposium, 2008, pp. 323–332.
[25] AMD, Cooln'quiet real world numbers, http://icrontic.com/article/socket_940_vs_939, 2004.
[26] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., New York, NY, USA, 1979.
[27] Amazon elastic compute cloud (Amazon EC2), 2012.
[28] Amazon elastic compute cloud (Amazon EC2) instance type, 2013.
[29] Grid5000, https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home, 2013.