A Pareto-based Metaheuristic for Scheduling HPC Applications on a Geographically Distributed Cloud Federation

Yacine Kessaci, Melab Nouredine, El-Ghazali Talbi

Cluster Computing journal manuscript No.
(will be inserted by the editor)

A Pareto-based Metaheuristic for Scheduling HPC Applications on a Geographically Distributed Cloud Federation

Yacine Kessaci · Nouredine Melab · El-Ghazali Talbi

Received: date / Accepted: date

Abstract Reducing energy consumption is an increasingly important issue in cloud computing, more specifically when dealing with High Performance Computing (HPC). Minimizing energy consumption can significantly reduce the amount of energy bills and then increase the provider's profit. In addition, the reduction of energy decreases greenhouse gas emissions. Therefore, many researches are carried out to develop new methods in order to make HPC applications consuming less energy. In this paper, we present a multi-objective genetic algorithm (MO-GA) that optimizes the energy consumption, CO_2 emissions and the generated profit of a geographically distributed cloud computing infrastructure. We also propose a greedy heuristic that aims to maximize the number of scheduled applications in order to compare it with the MO-GA. The two approaches have been experimented using realistic workload traces from Feitelson's PWA Parallel Workload Archive. The results show that MO-GA outperforms the greedy heuristic by a significant margin in terms of energy consumption and CO_2 emissions. In addition, MO-GA is also proved to be slightly better in terms of profit while scheduling more applications.

Keywords scheduling, cloud computing, green computing, resource allocation, multi-objective optimization, genetic algorithm

INRIA Lille Nord Europe - LIFL/CNRS UMR 8022 - Université Lille 1
40 avenue Halley, 59650 Villeneuve d'Ascq Cedex FRANCE.
E-mail: Yacine.Kessaci@lifl.fr, Nouredine.Melab@lifl.fr, El-Ghazali.Talbi@lifl.fr

1 Introduction

Cloud computing appears nowadays to be increasingly adopted in many areas. The field of high performance computing (HPC) does not derogate to this rule. However, computers use a significant and growing portion of energy in the world. Therefore, energy-aware computing is crucial for large-scale systems that consume considerable amount of energy. A recent study [16] shows that in 2005, the power used by servers represents about 0.6% of total U.S. electricity consumption. That proportion grows to 1.2% when cooling and auxiliary infrastructures are included. In the same year, the aggregate electricity bill for operating those servers and associated infrastructure was about $2.7 billions and $7.2 billions for the U.S. and the world, respectively. The total electricity consumed by servers doubled over the period 2000 to 2005 in worldwide and this increase was further confirmed in the last 5 years (2005-2010)[4].

On the other hand, green house gas emission is reaching a critical limit. A recent work [12] estimates that the global Information and Communications Technology (ICT) industry accounts for approximately 2% of global carbon dioxide emissions. This is equivalent to the amount emitted by the aviation. To face this phenomenon different governments are fixing limits to (ICT) industries.

Energy consumption has another drawback by affecting the profit of the providers. Indeed, according to Amazon's estimate [13], 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 all the issues cited before are somehow related and thus have to be dealt with simultaneously.

Unlike our work, most of existing works tackle the energy aware allocation problem focus on individual data centers or on centralized architectures like in [17, 28].Moreover, these works [14, 26], propose methods only for specific tightly coupled applications. In addition, the works dealing with the energy aware scheduling topic are either mono-objective or multi-objective. The former follows two approaches, heuristic ones [17] and genetic algorithm ones [20]. In the latter, we find also heuristics and genetic algorithms. The heuristics use either the lexicographic or the aggregation method to deal with an additional objective like in [18, 8], while the genetic algorithms treat the multi-objective issue with a Pareto approach as proposed in [21]. Other works in cloud computing focus only on profit maximization and do not pay attention to the energy consumption [19, 10, 30, 6].

Our work differs from the previous studies in plenty aspects. First, it deals with both computing and cooling energy consumptions in the energy model. It uses a multi-objective evolutionary algorithm in the meta-scheduler in order to do not favor any of the objectives. This allows one to obtain a Pareto set of solutions and show the trade-off between all the tackled criteria. Finally, the experiments in our work are realistic and performed on a long period of workloads composed of heterogenous HPC applications in order to avoid the tightly coupled applications issue.

In this paper, we propose a new Pareto resource allocation approach for clouds based on three criteria: energy, green house gas emission and profit. Indeed, as previously said a meta-scheduler that uses a multi-objective genetic algorithm (MO-GA) is proposed in order to find the best scheduling according to those three objectives. The main contribution of our approach is the benefits that the meta-scheduler can draw from the geographical distribution of the clouds to find the best meta-scheduling since energy, CO_2 and profit can be different over the world. In fact, each area in the world has characteristics, such as: temperature, electricity price, workloads, green house gas emissions rates, hardware specifications, etc.

Our approach also aims to give the best Quality of Service (QoS) to the clients by meeting the maximum application's deadlines. In addition, our approach satisfies also the provider by using a mechanism that gives him/her the ability to make a dynamic choice among the Pareto set of the proposed solutions according to his/her real time needs in order to improve the results.

This paper is an extended version of the work presented in [15]. Indeed, in this paper the related work, the considered model and the algorithm features are more detailed, a deeper evaluation process is applied and more experiments with new instances and different comparison aspects are realized.

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

2 Related Work

After a race to performance, utility and cloud computing paradigm are facing an energy problem. Hence, several works have been proposed in the field of the energy aware computing. However, most of those approaches tackle this topic by referring to single data center and focusing on scheduling dedicated applications. In [18, 23] for example a hardware technique (DVFS) is proposed, it consists of varying the CPU frequency in order to minimize the energy consumption. The drawback of this type of methods is the assumption that they make about a tight coupling between the tasks and the resources. Another way of reducing cloud computing energy footprints is proposed in [17]. This work uses the possibilities offered by the virtualization in order to apply a task consolidation through two heuristics in order to maximize the resource utilization. In [28] the author presents a reinforcement learning approach to deal with the optimization of two main aspects, performance and power consumption. All the previous presented works aim to reduce the energy consumption on single data center or on multiple servers geographically concentrated, except the work proposed in [22] which deals with energy consumption reduction in large-scale computational grids like Grid5000, by switching off idle nodes in a clever way.

Other approaches treat the economic side of cloud computing, like in [19], where two algorithms based on a pricing model are proposed. They both use processor sharing in order to balance between conflicting objectives (profit and resource utilization). In [6] Burge *et al* describe a method for heterogeneous machines that maximizes the profit by assigning the requests to the machines according to their energy cost. Other approaches based on genetic algorithms and dealing with profit are presented in [30] and [10]. In [10] a linear programming driven genetic algorithm is proposed. In fact, this work aims to give the best meta-scheduling in a utility grid based on the idea of minimizing the combined costs of all users in a coordinated way. Yu and Buyya in [30] present a genetic algorithm approach to address scheduling optimization problems in workflow

applications with two QoS constraints (deadline and budget).

All of the last presented approaches take into account the profit or the energy in their study but they do not consider the relationship between energy, green house gas emissions and profit. They also do not pay attention on how each one of those criteria can affect the others. The work presented by Garg *et al* in [11] deals with those points, by proposing a new energy model that includes gas emissions and pricing. Several heuristics are proposed to find a good tradeoff between the objectives. However, this approach is an aggregation of objectives (i.e. it can only optimize one objective at time).

Therefore, to deal with all the misses mentioned before, we propose a meta-scheduler using a multi-objective genetic algorithm to optimize the whole three objectives at the same time. In other words, our new approach provides a set of Pareto solutions (i.e. non-dominated solutions) rather than a single solution.

Table 1 summarizes and locates our approach among the other ones.

3 Distributed Cloud Scheduling Model

3.1 System model

Our cloud model is an Infrastructure As A Service (IAAS). More precisely, we are dealing with a two-tier architecture: on one side the distributed cloud provider and, on the other side, the clients. The clients have access to the cloud by requesting resources to the provider. The service proposed by the cloud provider in our approach is offering infrastructures to the clients in order to run their HPC applications. The role of this work is to help the provider to optimize a certain number of criteria while proposing its service. The model of our architecture is a cloud federation which is geographically distributed over the world inspired from the Open Cirrus project [24]. The originality of this approach is to propose a meta-scheduling algorithm that uses a multi-objective genetic algorithm in order to find the best meta-scheduling to the applications over the time. Three objectives are considered: energy, carbon emissions and profit. The client's QoS constraints include the execution time, the number of CPUs and the respect of the applications' deadlines. To meet those constraints, the meta-scheduler has to ask each cloud over the world about information concerning the CPU's states and their availability. In addition to optimizing the previously cited three objectives and thus helping the provider to maximize his profit, the meta-scheduler algorithm aims also to give the best Quality of Service (QoS) for the client by meeting the maximum applications deadlines and respecting model's constraints. The optimization of the objectives is due to the characteristics offered by the geographical distribution of the clouds (Cloud Federation). Indeed, the profit is related to the difference between the electricity prices over the world and the gas emissions due to the used methods in those places to produce the electricity power. This will generate different amounts of green house gas emissions from an area to another. The role of each third of our model is detailed in the following:

- **User side:** The requests submitted by the distributed cloud's users are HPC applications. This means that the service is computation-intensive. Hence, we do not pay attention in this work to data transfers. In our approach the clients submit HPC applications by informing the meta-scheduler about their execution time and the number of processors needed. The information about execution time is deductable by two factors. As a first factor we can take the real world submission. With this method the execution time represents the reservation time of the user. Hence, for his/her interest the client has to reserve enough time for his/her application, otherwise his/her application will be aborted. Therefore, the user has some time to overestimate the execution time of his/her request and pays for longer than the real execution time of his/her application. The second factor is prediction. Indeed, nowadays predicting an execution time of an application is starting to be possible by using benchmarks and historical data for instance. This last technique is the one that we use in our work. Concerning the deadlines, they are specified by the client and are represented as a strong constraint in our model. In other words, if the meta-scheduler is not capable to find a slot of time to satisfy the request by respecting the deadline, the reservation for the application will not happen. Each HPC application has to be hosted in one and only one data center. This constraint helps to respect the configuration of our cloud federation which is a loosely coupled cloud (i.e. no possibility of communication between the clouds). All the requests have the same priority (i.e. there is no preemption in our model). The only priority is the order of the request's arrival. Another reason of not dealing with the distribution of the applications is that in our work we focus on high level scheduling and thus, the distribution of the application's tasks is let to a lower level like in [18].
- **Provider side:** In our approach the provider is the owner of all the data centers over the world. For instance Amazon [3], one of the world leader in

Table 1 Classification of the related work.

Approach	Greenhouse gas emission/Energy consumption	Energy cost aware scheduling	Genetic algorithm	Pareto optimization schedulers
Rizvandi et al.[23]	yes	no	no	no
Lee and Zomaya[17]	yes	no	no	no
Tesauro et al.[28]	yes	no	no	no
Orgerie et al.[22]	yes	no	no	no
Lee et al[19]	no	no	no	no
Burge et al.[6]	yes	yes	no	no
Yu and Buyya[30]	no	no	yes	no
Garg et al.[10]	no	no	yes	no
Garg et al.[11]	yes	yes	no	no
Our work	yes	yes	yes	yes

Approach	HPC workload	Distributed data centers	Market-oriented schedulers
Rizvandi et al.[23]	no	no	no
Lee and Zomaya[17]	no	no	no
Tesauro et al.[28]	yes	no	no
Orgerie et al.[22]	yes	yes	no
Lee et al[19]	no	no	yes
Burge et al.[6]	yes	no	no
Yu and Buyya[30]	no	yes	yes
Garg et al.[10]	no	yes	yes
Garg et al.[11]	yes	yes	yes
Our work	yes	yes	yes

the field of cloud computing, has clusters deployed in three different continents, North America (USA: California and Virginia), Europe (Ireland) and Asia (Singapore). In our model after each request the meta-scheduler asks all the data centers about their states in order to choose the best possible scheduling. The information provided by the clouds is the number of processors available during the requested time (i.e. the users' request is compared with the available slots of each cloud). Each slot is a period of time that satisfies the request for one processor. The cloud has to provide a set of slots equal to the specified number of processors needed by the client. Each time a cloud satisfies this condition and other specifications, it will be chosen by the meta-scheduler and its state will be updated. Those other specifications are given by the local scheduler of each cloud to the meta-scheduler helping it to find the best meta-scheduling. Those specific cloud information are the execution price, the carbon emission rate, the Coefficient of Performance (COP), the electricity price, the CPU power, the CPU frequency and the number of CPUs. All the processors within a cloud are homogeneous, this can be justified by using virtualization techniques such as VMware Fusion, Xen and Linux KVM. The gas emission rates are provided by multiple agencies such as ADEME (Agence de l'Environnement et de la Maîtrise de l'Énergie) in France and EIA (Energy Information Administration) in the USA.

3.2 Energy model

The energy consumption of a data center (cloud) results from IT equipments (network, storage and computing) and auxiliary equipments (lighting, cooling ...). In our work, we do not consider lighting consumption among the auxiliary equipment since its impact is negligible. Regarding IT equipments, we deal only with computing energy consumption. Indeed, since our approach is focused on HPC applications, the largest amount of energy is consumed by the intensive computation. Our approach also does not pay attention to how the energy is optimized within the cloud but between the federated clouds.

Our energy model is derived from the power consumption model in Complementary Metal-Oxide Semiconductor (CMOS) logic circuits. The power consumption of a CMOS-based microprocessor is defined as the summation of capacitive, short-circuit and leakage power. The capacitive power (dynamic power dissipation) is the most significant factor of the power consumption. The power dissipation P is defined as:

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

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 resulting from switching between a voltage to another. The power dissipation is not influent in our study since we do not use the Dynamic Voltage Scaling (DVS) method to be able to perform voltage switching. Notice that A and C are constant values, let α be their product. The second part of the equation represents the static consumption, this value is also constant, let it be β. In CMOS processors the voltage can be expressed as a linear function of frequency, $V^2 f$ is replaced by f^3. The new equation becomes:

$$P = \alpha f^3 + \beta \quad (2)$$

In addition, another source of energy consumption needs to be taken into account. In fact, the energy used for cooling each cloud's data center is consequent and has to be integrated in our energy model. Energy dedicated to cooling is tightly related to the geographical area where the data center is situated since the temperature changes from an area to another. To compute cooling energy amount, each data center has a coefficient called COP which represents the ratio between the energy dedicated for the execution of the request and the energy used for cooling the system. The meta-scheduler is informed about the COP value by each cloud's local scheduler while submitting the first request or if COP value changes during the time. By using COP the meta-scheduler is able to deduce the energy consumed by each data center for cooling their devices. This is given by Equation (3).

$$E^h = E^c / COP \quad (3)$$

where E^h represents the total energy consumed for cooling the data center and E^c represents the energy used by the CPUs.

The pricing model is directly related to the energy model, since the less energy the provider consumes the more important his/her profits will be. However, another parameter affects the result of the profit: electricity price. Indeed, the electricity price changes from a site to another. The profit is then the difference between the fixed price that the client pays and the price that the provider has to pay for his/her electricity consumption (Equation (4)).

$$Profit = pr^u - pr^e \quad (4)$$

where pr^u is the price that the user pays for the service and pr^e is the electricity price that the provider pays to provide the service.

3.3 Problem formulation

In our cloud model, we deal with a two-tier architecture. The first tier is the cloud provider which has N clouds geographically distributed over different areas in the world. The second tier represents clients with J HPC applications that have to be executed on the clouds. The problem consists of scheduling J applications on N clouds. We know that the task scheduling problem in general is NP-hard [9]. Therefore our multi-objective scheduling problem is NP-hard as well. Thus, a metaheuristic algorithm appears to be the most appropriate approach to be adopted.

In our formulation the provider has to pay the execution price of the used cloud i, this price is the result of the electricity consumption during the computation and is noted p_i^e (\$/kW h). According to p_i^e the provider fixes a price for the clients. We designate the fixed client price per hour by p^c (\$/CPU/hour). The CO_2 amount of each cloud i is calculated from a ratio noted $r_i^{CO_2}$. This ratio is an average value that varies according to the way the cloud's electricity is produced (i.e. type of energy used for the electric power generation: fuel, water, nuclear, wind ...).

During the scheduling process, the user submits a request for an HPC application j. A request is defined by a triplet (e_j, n_j, d_j), all the triplet's information are given by the user during the reservation, except the starting time of the application t_j which is deduced from the submission time. The elements of the triplet (e_j, n_j, d_j) represent for e_j, the execution time (reservation time) of the application, for n_j, the number of processors needed by the user for his application and finally d_j, for the deadline after what the application will be considered as failed. Our triplet is inspired from Amazon EC2 [3] which requires from the user to provide the duration time of his/her application. Thus, the user has sometimes to pay for a longer reservation time to ensure the completion of his/her application, even if this application finishes before the end of the reservation time.

In the following, we present the mathematical formalism of our problem and the used functions for computing the fitness of each candidate solution (scheduling).

- Energy consumption of the CPUs is given by:

$$E_{ij}^c = (\alpha_i (f_{ij}^3) + \beta_i) \times n_j e_j \quad (5)$$

- From Equation (5) and the Coefficient of Performance (COP) the total consumed energy is deduced as:

$$E_{ij} = \frac{COP_i + 1}{COP_i} \times E_{ij}^c \quad (6)$$

- The total carbon emission is given by:

$$(CO_2 E)_{ij} = r_i^{CO_2} \times E_{ij} \quad (7)$$

– The profit is given by:

$$(Profit)_{ij} = n_j e_j p^c - C^e_{ij} \quad (8)$$

where the client's bill for the execution of an application j is the product between the fixed price unit p^c, the number n_j of used processors by the application j and the execution time e_j of the application j. C^e_{ij} is the price that the provider has to pay for the used resources in the cloud i for executing the application j.

Equation (6) uses COP in order to add the cooling energy to the CPU energy. Indeed, $(COP + 1/COP) \times E^c$ equals $E^h + E^c$ can be proven easily in the following:

From Equation (3)

$$\begin{aligned} \mathrm{E} = \quad & E^c + E^h = E^c + \tfrac{E^c}{COP} \\ = \quad & E^c \times (1 + \tfrac{1}{COP}) = \tfrac{COP+1}{COP} \times E^c \end{aligned} \quad (9)$$

The objective functions of our approach are formulated as follows:

$$Minimizing\,the\,energy\,consumption = \sum_i^N \sum_j^J (E)_{ij} \quad (10)$$

$$Minimizing\,Carbon\,Emission = \sum_i^N \sum_j^J (CO_2E)_{ij} \quad (11)$$

$$Maximizing\ Profit = \sum_i^N \sum_j^J (Profit)_{ij} \quad (12)$$

with the following constraints:

– The application j has to finish before d_j otherwise the scheduling is rejected,
– Each application j can be assigned to one and only one cloud j.

The objective functions aim respectively to minimize the energy consumed by the entire distributed cloud for the Equation (10), to reduce the distributed cloud's carbon emissions for the Equation (11) and to maximize provider's profit for the Equation (12). The Equation (11) could be wrongly considered similar to the Equation (10), but they are different and contradictory. Indeed, the carbon ratio $r_i^{CO_2}$ has no relationship with the energy consumption and thus a cloud with a good energy features is not necessary good for the CO_2 emissions. The correlation coefficient between those two objectives on a sample of 1000 solutions is 0.57.

4 MO-GA for Meta-scheduling

In this section, we describe in details the steps of our approach and thus the multi-objective genetic algorithm (MO-GA) proposed in our study.

4.1 Multi-objective combinatorial optimization

A multi-objective optimization problem (MOP) consists generally in optimizing a vector of nb_{obj} objective functions $F(x) = (f_1(x), ..., f_{nb_{obj}}(x))$, where x is an d-dimensional decision vector $x = (x_1, ..., x_d)$ from some universe called decision space. The space the objective vector belongs to is called the objective space. F can be defined as a cost function from the decision space to the objective space that evaluates the quality of each solution $(x_1, ..., x_d)$ by assigning it an objective vector $(y_1, ..., y_{nb_{obj}})$, called the fitness. While single-objective optimization problems have a unique optimal solution, a MOP may have a set of solutions known as the Pareto optimal set. The image of this set in the objective space is denoted as the Pareto front. For minimization problems, the Pareto concepts of MOPs are defined as follows (for maximization problems the definitions are similar):

– *Pareto dominance*: An objective vector y^1 dominates another objective vector y^2 if no component of y^2 is smaller than the corresponding component of y^1, and at least one component of y^2 is greater than its correspondent in y^1 i.e.:

$$\begin{cases} \forall i \in [1..nb_{obj}],\ y^1_i \leq y^2_i \\ \exists j \in [1..nb_{obj}],\ y^1_j < y^2_j. \end{cases} \quad (13)$$

– *Pareto optimality*: A solution x of the decision space is Pareto optimal if there is no solution x' in the decision space for which $F(x')$ dominates $F(x)$.
– *Pareto optimal set*: For a MOP, the Pareto optimal set is the set of Pareto optimal solutions.
– *Pareto front*: For a MOP, the Pareto front is the image of the Pareto optimal set in the objective space.

Graphically, a solution x is Pareto optimal if there is no other solution x'such that the point $F(x')$ is in the dominance cone of $F(x)$. This dominance cone is the box defined by $F(x)$, its projections on the axes and the origin (see Fig. 5).

4.2 Problem encoding

In order to formulate our problem without overriding the previous constraints (i.e. finishing the application before its deadline and scheduling each application on one and only one cloud), we propose an encoding for the MO-GA individuals (see Fig. 1).

Fig. 1 represents one possible scheduling among plenty that proposes the genetic algorithm. This scheduling is the result of processing a pool of requests arrived during the last waiting time period presented later and called *scheduling cycle*. In the proposed example we identify

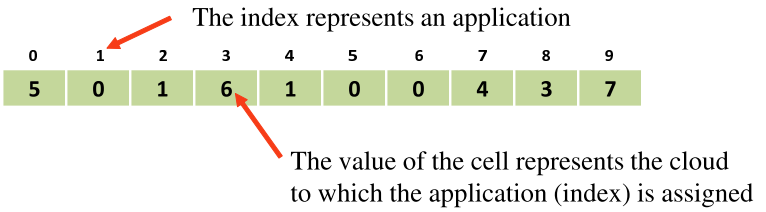

Fig. 1 A representation of a solution in the meta-scheduling problem (individual in MO-GA's population).

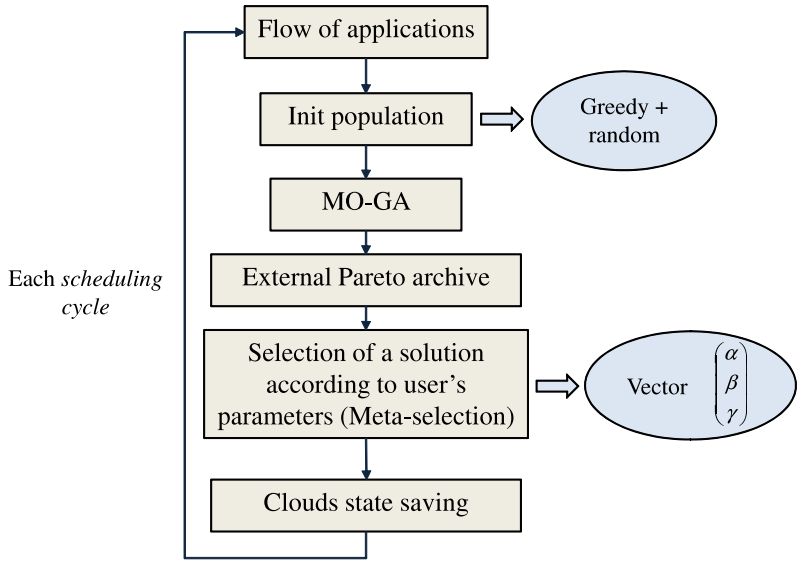

Fig. 2 A flowchart representing the meta-scheduling algorithm's steps.

three major specifications. The indexes of the table depict the applications that are scheduled, the number in each table cell identifies the cloud on which the application is allocated. In other words, the first cell represents the first application of the pool that is currently handled by the MO-GA, in this case this application is allocated to the cloud 5. The second application is allocated to the cloud 0 and so on. This encoding informs about the number of applications contained by the pool, which is 10 in our example. This encoding helps us also to deal with the characteristics of our problem. Indeed, it allows to schedule all the applications of the pool by assigning each one to only one cloud. But a cloud is able to handle more than one application. Note that not all the clouds are necessarily used in each solution. The last constraint of our model which can not be handled by the proposed encoding is the deadline constraint. We deal with this constraint in the algorithm by rejecting the requests (applications) that do not respect the deadlines. In other words, all the requests that compose each individual of the MO-GA at each processing cycle, satisfy all the requirements on the current federation cloud state (i.e. it exists at least one cloud in the federation, at the current time, which can handle the request in term of number of processors and respect of the deadline).

4.3 Population initialization

The initialization of the population in a genetic algorithm is an important phase. In fact, this step affects the quality of the future results. The initialization of the population is done according to a combination of two methods. The first method rely on a greedy algorithm and the second is a random initialization method. The initialization is decomposed into three steps as follows:

- The first step reads the pool of applications with the greedy method.
- The second step initializes either one or two elements of the population by the result of the first step.
- The third step initializes the rest of the population with a random method.

The greedy method read the application that arrive during the scheduling cycle and allocate them to the clouds. The allocation follows the order of the applications arrival with as only constraint, meeting the deadline of each application. Each application that can

not be allocated by the greedy method is considered as failed and will not be a part of any of the future scheduling pools of applications. This first step of the initialization process helps to avoid having a total failed scheduling because of only one application that can not meet its deadline. In other words, this step makes sure that there is at least one feasible solution (scheduling) in the algorithm population. Otherwise, the genetic algorithm rejects a big number of applications among the entire pool only because of one unmet deadline's application. Having this greedy method coupled to the random method to initialize the rest of the population helps to add diversity to the initial population and thus, do not bias the search of MO-GA. The size of the pool of applications is equal to all the applications arrived during the *scheduling cycle* minus the ones eliminated in the initialization phase. The ratio between the individuals initialized by the greedy algorithm and those initialized by the random method is 1/15 (i.e. 14 random for each 1 greedy).

4.4 Meta-scheduling algorithm steps

Before each scheduling, the meta-scheduler waits a fixed period called *scheduling cycle*. This period helps to gather a pool of applications in order to have a larger choice and thus, optimize the scheduling. Once this phase done the pool is managed by the MO-GA to find the best schedulings possible over the different clouds which compose the distributed cloud. The result of the execution is stored as a Pareto archive. Once the set of Pareto solutions (schedulings) is proposed, the algorithm chooses one scheduling according to the user's (provider) choice. The chosen solution from the Pareto set is used as a state for the cloud federation. This state will be a basis from which the next iteration of the algorithm will make another execution on a new pool of applications. The algorithm keeps iterating and proposes schedulings for each new pool of applications (see Fig. 2).

4.5 Genetic algorithm

Genetic Algorithms (GAs) are meta-heuristics based on the iterative application of stochastic operators on a population of candidate solutions. In the Pareto-oriented multi-objective context, the structure of the GA remains almost the same as in the mono-objective context. However, some adaptations are required like in our MO-GA.

The MO-GA starts by initializing the population as indicated in Section 4.3. This population like said before is used to generate offsprings using the mutation and crossover operators. Each time a modification is performed by those operators on each individual, an evaluation operator (fitness) is called to evaluate the offsprings. The fitness in MO-GA is deduced from the energy consumption, CO_2 emissions and the generated profit of each scheduling (solution). The method used in the MO-GA to rank the individuals of the population, because of the multi-objective context, is the dominance depth fitness assignment. Hence, only the individuals (solutions) with the best rank are stored in the Pareto archive. This archive contains the different non-dominated solutions generated through the generations. The next step of the MO-GA is the selection process. It is based on two major mechanisms: elitism and crowding. They allow respectively the convergence of the evolution process to the best Pareto front and maintaining some diversity of the potential solutions. The elitism mechanism makes use of the population in the Pareto archive. Such an archive is updated at each generation and used by the selection process. Indeed, the individuals on which the variation operators are applied are first, selected according to their rank using the non-dominance concept, either from the Pareto archive, from the population or from both of them. In the second step, the crowding process gets involved, it maintains diversity in the solutions by ranking again the individuals according to the crowding distance. This is done on the basis of the similarity degree of each individual compared to the others. The similarity (diversity) in crowding is defined as the circumference of the rectangle defined by its left and right neighbors, and infinity if there is no neighbor. These mechanisms are the same as the ones used in the NSGA-II algorithm. More details about these techniques are given in [27].

When new solutions (offsprings) are generated a replacement of the old solutions is necessary in order to keep constant the number of individuals in the population. The selection operator in the replacement process is based on a tournament strategy. Tournament selection consists in randomly selecting k individuals, where k is the size of the tournament group. The replacement of the old solutions follows an elitist strategy where the worst individuals of the population are replaced by the new ones (offsprings). The algorithm stops when no improvement on the best solutions is performed after a fixed number of generations. Once this number of iteration reached, the external Pareto archive of the meta-scheduler is updated by the last Pareto archive of the MO-GA.

Regarding the principle of the stochastic variation operators of MO-GA we have: in one hand, the mutation operator which is conventional. Indeed, the operator chooses randomly two integers i and j such that

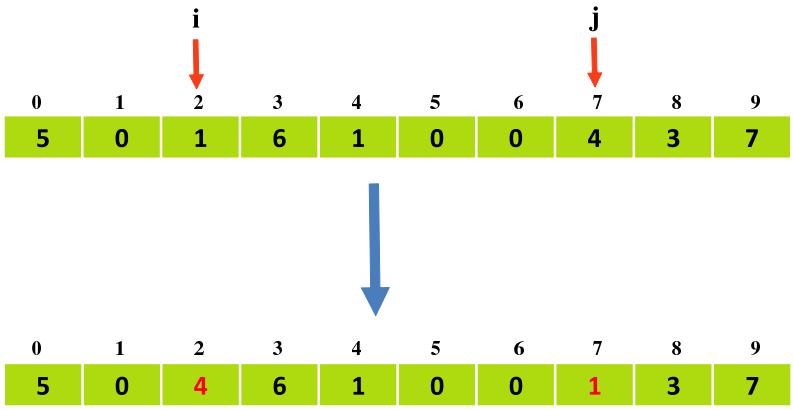

Fig. 3 The mutation operator mechanism used in MO-GA to reassign two applications by swapping two clouds.

$1 \leq i < j \leq n$. Then, the operator swaps the two applications i and j like in Fig. 3. In the other hand, the crossover operator which uses two solutions $s1$ and $s2$ to generate two new solutions $s1'$ and $s2'$. The operator picks also two integers on each solution to make the crossover. The full mechanism is explained bellow and illustrated in Fig. 4. However, these operations are done only if the number of the scheduled applications is greater than 2 for the mutation and than 3 for the crossover. Indeed, when no operator can be applied (i.e. only one application to schedule), the diversity is obtained from the number of the individuals of the population resulting from the initialization.

To generate $s1'$, the operator:

- considers $s1$ as the first parent and $s2$ as the second parent.
- randomly selects two integers i and j such that $1 \leq i < j \leq N$.
- copies in $s1'$ all tasks of $s1$ located before i or after j. These tasks are copied according to their positions ($s1'_k = s1_k$ if $k < i$ or $k > j$).
- copies in a solution s all tasks of $s2$ that are not yet in $s1'$. Thus, the new solution s contains $(j - i + 1)$ tasks. The first task is at position 1 and the last task at the position $(j - i + 1)$.
- and finally, copies all the tasks of s to the positions of $s1'$ located between i and j ($s1'_k = s_{k-i+1}$ for all $i \leq k \leq j$).

The solution $s2'$ is generated using the same method by considering $s2$ as the first parent and $s1$ as the second parent.

4.6 Cloud federation state selection

The results obtained using MO-GA are stored in a Pareto archive. Hence, starting the process of a new pool of application with several solutions from the Pareto set becomes difficult. Therefore, in our meta-scheduling algorithm there is a meta-selection step which comes right after the end of the MO-GA. This step aims to pick up a solution among the external Pareto archive in order to set the distributed cloud state. This state will be the starting point from which the next execution of MO-GA will schedule a new pool of applications. The idea behind choosing a Pareto approach in our work is to propose to the provider as many compromise solutions as possible. Each one of these solutions is better than the other regarding a certain objective. The mechanism of meta-selection of the solution can be seen in different ways. The first and trivial mechanism is a manual choice done at each step by the provider according to his/her choices. The second one is a decision making algorithm that makes the adequate choice favoring the objectives to promote. Finally, our solution which uses a vector as an input parameter in order to automate the progression of the experimentations. Our vector parameter is a three dimensional vector. Indeed, since we deal with three objectives each dimension represents a weighting for a particular objective. In the meta-selection state step, the vector has a direction on which it points to. This direction is set by the provider. The solution that is the nearest to the vector's direction is the one which is chosen among the others in the Pareto set. In Fig. 5 we give an example with three two-dimensional vectors. In Fig. 6 we give an example of transition from an old state to a new one. The example concerns a four processors data center within a cloud federation where the applications are represented by Ai and the processors by Pj.

5 Experiments

This section presents the results obtained from our comparative experimental study. The experiments aim to

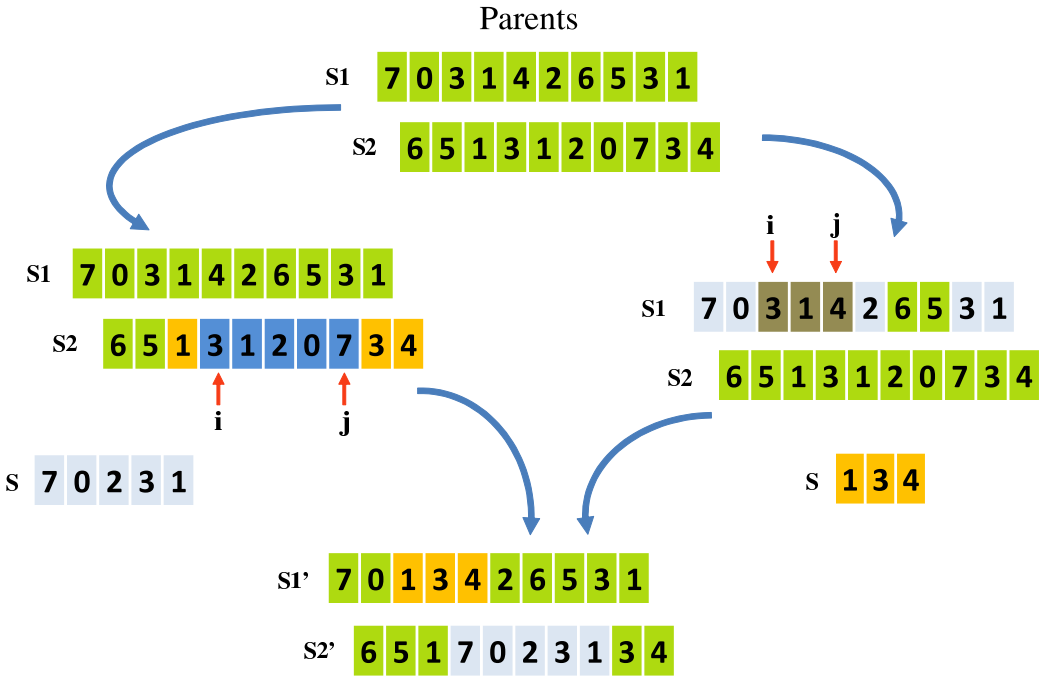

Fig. 4 The crossover operator mechanism used in MO-GA between two parent solutions $s1$ and $s2$ to generate two offspring solutions $s1'$ and $s2'$.

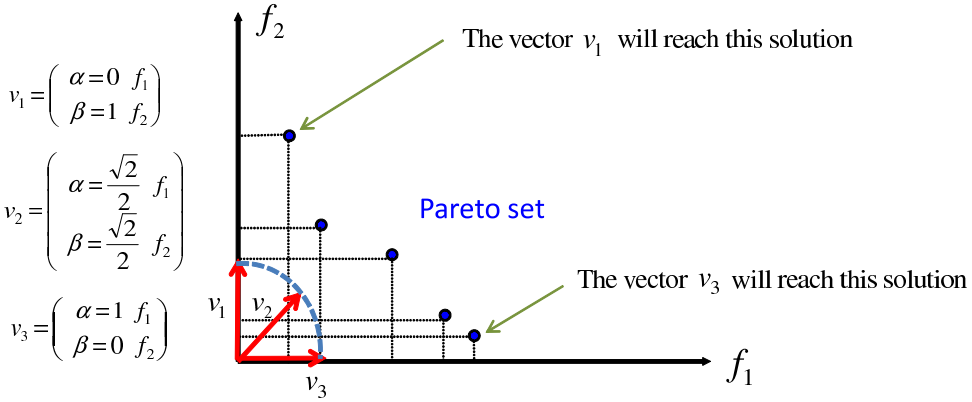

Fig. 5 The vector meta-selection mechanism applied to a bi-objective Pareto set in order to choose a particular solution.

demonstrate and evaluate the contribution of the multi-objective evolutionary approach with different meta-selection policies. It also aims to compare the obtained results of the MO-GA based meta-scheduler to a maximum application scheduling heuristic and to a random approach.

5.1 Experimental settings

The experimental settings concern both sides of our model, client side with its applications and provider side with the hardware configuration of the distributed cloud.

– **Application settings:** Since our approach deals with HPC applications, we use realistic workloads traces from Feitelson's Parallel Workload Archive (PWA) [7]. The workload traces stretch over a period of five months of applications (January 2007 to June 2007) for the first instance which uses the traces of the Lawrence Livermore National Laboratory (LLNL) from the Thunder cluster, and for a duration of two months (June 2010 to August 2010) for the RICC (RIKEN Integrated Cluster of Clusters) instance. We used those two traces because of their high rate of resources utilization 87.6% for the first and 87.2% for the second. This helps to simulate a heavy workload scenario. The reason why we choose the traces between June 2010 to August 2010 in the RICC instance is because of the high utilization rates and the offered load that offers this

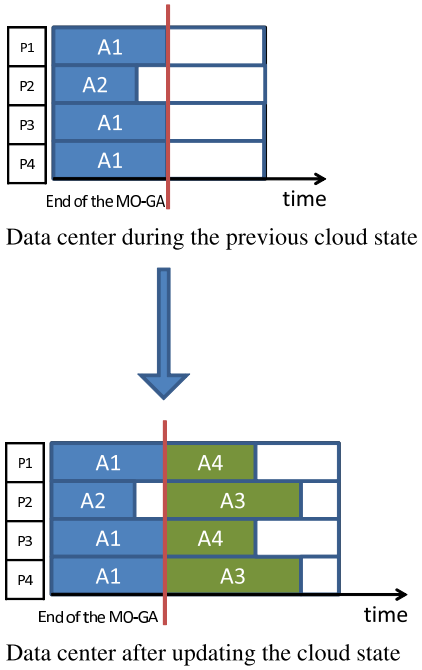

Fig. 6 The cloud state transition within a data center after the end of the MO-GA execution.

period. The information that we extract from both instances are the submit time, the execution time and the number of required processors. The traces have no information about the applications deadlines. We used the method presented in [29] to generate synthetically the deadlines for the needs of our experiments. The applications are classified into two classes named High Urgency (HU) and Low Urgency (LU). The generation of the deadlines of each class is performed according to a normal distribution. In order to have a distribution in both HU and LU classes we used a bimodal distribution in which, 80% of the generated values belong to LU and 20% to HU. The obtained results from this generation represent the ratio between $deadline_j/runtime_j$ of an application j. The application's deadline is deduced from such ratio and the execution time of the application. The used parameters for the bimodal distribution have in both classes a variance of 2, and a mean value of 12 for the class LU and 4 for the class HU. In other words, a HU application has three times less time on average to finish its execution than LU application. The HU and LU applications are distributed randomly in the sequence of the applications arrival.

– **Cloud federation settings:** In our approach we use 8 clouds geographically distributed with the same specifications as in [11]. The COP of each cloud is given by a uniform distribution between [0.6,3.5] as indicated in [25]. Table 2 shows the characteristics of the clouds which compose the cloud federation. The electricity prices and carbon emission rates are taken from respectively US Energy Information Administration (EIA) report [2] and US Department of Energy (DOE) [1]. Since we are dealing with a meta-scheduler we do not use energy reducing techniques within the clouds (data centers). Hence, the optimal frequencies of the processors in the clouds are not used.

5.2 Algorithm parameters

In our experiments we use some parameters such as the meta-selection state vector, the arrival rate of applications, the client execution price and the scheduling cycle. The meta-selection state vector presented in Section 4.6 is used in order to make the suitable choice while picking a solution in the external Pareto set and let the experiments continue from a pool of applications to another. We performed experiments with four different vectors. The first vector does not favor any of the three objectives, the second advantages the energy criterion, the third is more for the CO_2 criterion and the last one gives the maximum favors to the profit criterion. Regarding the arrival rate variation, we vary the original workload by changing in each arrival rate the submit time of the applications. We used four arrival rates in our experiments (Low, Medium, High and Very high). Each move from an arrival rate to another represents ten times more applications arrival during the same period of time. In other words, each time we

Table 2 Characteristics of the clouds which compose the cloud federation.

Location	COP rate	CO_2 rate (kg/kW h)	Electricity price ($/kW h)	CPU: α	CPU: β	Max frequency	Optimum frequency	Number of CPUs
New York, USA	3.052	0.389	0.15	65	7.5	1.8	1.630324	2050
Pennsylvania, USA	1.691	0.574	0.09	75	5	1.8	1.8	2600
California, USA	2.196	0.275	0.13	60	60	2.4	0.793701	650
Ohio, USA	1.270	0.817	0.09	75	5.2	2.4	1.93201	540
North Carolina, USA	1.843	0.563	0.07	90	4.5	3.0	2.154435	600
Texas, USA	1.608	0.664	0.1	105	6.5	3.0	2.00639	350
France	0.915	0.083	0.17	90	4.0	3.2	2.240702	200
Australia	3.099	0.924	0.11	105	4.4	3.2	2.285084	250

switch from an arrival rate to another we divide the submission time by 10. Thus, by shortening the submit time of the applications we increase the workload. The client price is fixed as the twice of the average energy cost of the clouds in the federation. Scheduling cycle in our algorithm is set to 50s. Table 3 summarizes the parameters used in our experiments.

Table 3 Experimental parameters.

Parameter	Value
Total number of applications	119849 + 115855
State selection vector	$\left(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right)$ (1,0,0) (0,1,0) (0,0,1)
Arrival rate	Low, Medium High, Very high
Client execution price	$0.40/CPU/h
Scheduling cycle	50s

Table 4 MO-GA parameters.

Parameter	Value
Population size	30
Number of generations	2000
Crossover rate	1
Mutation rate	0.35
Tournament group size	2

5.3 Maximum applications scheduling heuristic and random approach

To the best of our knowledge, there is no approaches dealing with the problematic of a Pareto multi-objective meta-scheduling on a geographically distributed cloud infrastructure. Therefore, we present briefly a heuristic and the random approach that we have used to compare our evolutionary approach to. The heuristic aims to assign the applications according to their arrival rate (First fit). After the *scheduling cycle* and the arrival of a new pool, the heuristic aims to maximize the QoS of the client (the number of scheduled applications). To do so, it chooses randomly a cloud among the federation and fills it by the maximum number of requests, when the cloud could not support the application requirements the heuristics chooses another cloud and so on until it finds a cloud who satisfies the requirements. If no cloud is found to handle the client request, the request is rejected. The objective of this heuristic is to avoid both rejecting requests and introducing free slots inside each cloud. Indeed, reducing the number of slots and maximizing the usage of each cloud minimize the total energy consumption by saving the cooling energy of all the unused cloud.

The random approach is based as its name indicates on a random assignment of the applications on the clouds composing the federation according to the arrival rate in a multi-objective way. Indeed, the obtained assignment after the *scheduling cycle* and the arrival of a new pool, is evaluated according to the number of scheduled applications and the value that this scheduling obtains in the three objectives. The final result for each instance over the whole workload is the sum of the results obtained during each *scheduling cycle*.

5.4 Performance evaluation

As said before, no previous approach deals with a Pareto multi-objective genetic algorithm for a distributed cloud meta-scheduler. Thus, we perform a bench of experiments with different parameters. In addition of optimizing the three objectives, the approach has first to satisfy the maximum number of clients QoS. In other words, the meta-scheduler has to handle the maximum number of applications. A comparison between our approach, a maximum applications scheduling heuristic and a random based approach, both presented in Section 5.3 seems to be the best choice to evaluate our work.

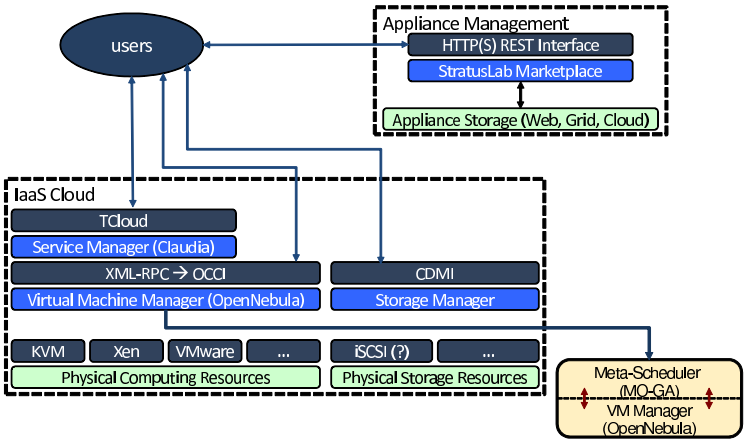

Fig. 7 How to integrate the MO-GA Meta-Scheduler in the cloud distribution StratusLab.

In order to switch from a cloud state to another we used 4 different vectors (see Table 3). These vectors help through their coordinates to choose the type of the solution (scheduling) that will be used for the cloud switching state. The vectors can help also to extract the most suitable solution among the Pareto set for a given objective and to compare our Pareto approach to a non Pareto approach. The results are presented in Tables 5 to 12.

The results of each instance (LLNL and RICC), for each arrival rate and for each meta-selection vector configuration of the MO-GA have been deduced from 30 independent runs. Besides, both the random approach and the heuristic have a part of randomness in their implementation. Therefore, the related results of both of those algorithms are deduced also respectively from 80 and 30 independent runs. The random part of the heuristic concerns only the selection cloud phase. In addition, the drawn values on the presented results are the medians of the results samples. Indeed, because of the non-normality of the distributions of the results through the different runs, and in order to be able to properly compare those values, we had to use the medians instead of the statistical averages. The detailed improvement rates of each objective in the comparison done between our approach and the maximum applications scheduling heuristic are presented in the Tables 15 and 16.

Experiments show that MO-GA has different behaviors according to the vector settings. Indeed, when set to *Average*, the meta-selection vector helps to have a constant progression in the results according to the different arrival rates in both instances (LLNL and RICC) and offers a large range of values on all the objectives (see Table 11 and Table 12). We deduce from this vector setting that it helps the provider to control the progression of the results over the different arrival rates, we also notice that the more the application rate is high the worse are the results and the higher the number of failed applications is. In addition, since this vector policy does not favor any objective, we obtain results that are less efficient compared to other vector orientations. On the other hand, the vector orientations that favor a specific objective obtain a significant improvement on this objective. Moreover, this improvement of solution quality concerns more the *Low* and *Medium* arrival rates than the *High* and *Very high* arrival rates. In other words, the improvement of the objective is the overall best compared to the other different orientation policies only for the *Low* and *Medium* arrival rates. For the other heavier arrival rates (*High* and *Very high*) the obtained results are good but they are not always the best values for the favored objective. The best value for a given objective for those kind of arrival rates is obtained with another orientation vector. This phenomenon can be explained by the local optima. Hence, when the provider keeps favoring the same objective during the arrival of a huge number of requests, all the clouds which can satisfy those requests by advantaging the considered objective become saturated and busy at the same time. This will conduce the future incoming applications to be assigned on clouds with worse specifications, which could be not interesting for optimizing the considered objective. An example of this observation is drawn in the instance LLNL for the oriented energy table Table 5. We obtain in that table for a *Very high* arrival rate better CO_2 emissions than in Table 9 where the vector is favoring the CO_2 criterion.

We notice this behavior more often in the experiments using the RICC instance. This is caused by the high utilization rate that proposes the RICC interval on which we conduct our experiments (i.e. about the same number of applications in both instances, despite a longer time interval in the LLNL instance -6 months-

compared to the interval of the RICC instance -2 months-). Indeed, in Table 8 which favors the profit, the CO_2 emissions are lower than in Table 10 which favors CO_2, always regarding the very high arrival rates. The same behavior is noticed in Table 8 which is Profit oriented, where the earned profit is lower for the *Very high* arrival rates than in Table 10 which favors CO_2. This is due to the fact that changing the orientation of the vector helps the algorithm to extract it self from a local optima when the clouds are saturated for a specific objective. It has the same effect as a kick move in a single based meta-heuristic like ILS (Iterative Local Search). We can conclude that ideal provider's behavior is to keep the vector orientation that favors the most wanted objective to be optimized only for the *Low* and *Medium* arrival rates. However, a more flexible orientation vector is suitable for the *High* and *Very high* arrival rates, by changing the orientation according to the real time algorithm behavior and to the targeted objective.

The comparison of the heuristic with the meta- scheduler MO-GA was done with an *average* orientation vector for the MO-GA, to be as fair as possible and do not favor any of the criteria. The obtained results over the different arrival rates on the LLNL instance show an improvement of 26% for the energy objective, 25.9% for the CO_2 objective and 1.8% for the profit while scheduling 2.2% more requests. For the second instance RICC, the results show an improvement of 29.4% for the energy consumption, 26.3% for the CO_2 emissions and 3.6% for the profit while scheduling 3% less applications. We notice that the improvement is more significant for RICC instance than for LLNL, this is due to the density proposed by the short interval of the RICC instance compared to LLNL interval (3 times shorter). This density highlights more the advantage of MO-GA compared to the heuristic, than on a longer well-balanced instance like LLNL. Furthermore, the detail of the improvement for each instance and for each arrival rate are presented in Table 15 and Table 16. Thus, the values in Table 15 for the LLNL Thunder instance show an improvement not matter the arrival rates, of the results obtained by the MO-GA meta-scheduler compared to the maximum applications scheduling heuristic. However, the improvement decreases according to the arrival rate increase. The best improvement of 51% concerns the CO_2 emission reduction for the *Low* arrival rate. Regarding the RICC instance results in Table 16, the improvement concerns all the arrival rates except the *Very high* arrival rate. Indeed, this deterioration is explained by the local optima phenomenon. When a high rate of application arrives, the MO-GA tends to optimize the criteria for this only applications' arrival regardless the next application arrivals. In fact, because of the real time arrival, the MO-GA meta-scheduler ignores their existence. However, on the other side, the heuristic which does not saturate the good resources because of a less optimal solution can benefit from those resources later during the next applications' arrivals and obtain therefore better final results. Same explanation goes for the increase in the failed application rate for the heavy arrival rates in the MO-GA. Moreover, the best improvement rate for the RICC instance obtained by MO-GA, concerns the energy reduction, by up to 55% compared to the heuristic for the *Low* arrival rate.

Concerning the time consumption of MO-GA, the results show that the heuristic gives results faster than MO-GA. However, that does not give any speed up to the algorithm during a real meta-scheduling. Indeed, between each processing, there is a waiting time *scheduling cycle*, where the algorithm waits for gathering a new pool of requests. The longest time taken by the meta-scheduler driven by MO-GA to treat 6 months of application requests for the LLNL instance, without counting the waiting time at each *scheduling cycle* (50 seconds), is roughly 19 hours and 40 minutes, while scheduling the 2 months requests of the RICC instance is done in less than 4 hours 42 minutes. We can deduce then that the MO-GA's processing time is covered by the *scheduling cycle* time, and that each pool scheduling is performed in less than 50 seconds.

The experiments of the random algorithm offer for both instances (LLNL and RICC) poor results. This approach does not optimize the client's QoS and therefore rejects a lot of feasible requests because of their random assignment on the clouds. For the *Very high* arrival rates it does not even give any results, and rejects all the requests, whether for the instance LLNL or for RICC (see random part in Table 5 and Table 6).

6 Conclusion

In this paper, we have presented a new meta-scheduler using a multi-objective genetic algorithm to minimize energy consumption, gas emission and maximize the profit while respecting applications' deadlines. The energy saving of our approach exploits the geographical distribution of the clouds that compose the cloud federation. Our work is considered as an optimization multi-objective method with a Pareto approach.

Our new approach has been evaluated using realistic workload traces of different instances from Feitelson's Parallel Workload Archive (PWA) [7]. Experiments show that our multi-objective GA improves on average the results obtained by the heuristic particularly in reducing the energy consumption. Indeed, the

energy consumption is reduced by up to **29.4 %**, the CO_2 emission by up to **26.3 %** and the profit is maximized by up to **3.6%**. In addition, our approach schedules on average the same number of applications than the heuristic that maximizes the number of scheduled applications. Therefore, one of the main perspectives of the work presented in this paper is to determine on one hand a way to minimize more the energy consumption by using DVS within the cloud's data centers, and on the other hand to modify the model by allowing delays for the applications by introducing a new pricing model with penalties. In addition, we can also imagine a dynamic meta-scheduler which will reassign applications during a scheduling phase on different clouds to optimize energy and/or profit. However, this will depend on the flexibility, the data transfer cost and the CPU time complexity of the applications since we deal with HPC applications.

Regarding the application of our work in practice, we are planning to collaborate with the StratusLab project [5]. Therefore, our MO-GA based meta-scheduler will take place as part of the modules that compose StratusLab (see Fig. 7). In fact, our approach will be integrated within the VM manager (OpenNebula) part of StratusLab, more specifically in its scheduling part to provide smarter assignments. Our approach will help to make an optimum use of the geographically distributed cloud offered by StratusLab through the EGI grid infrastructure. Thus, we will give the opportunity to exploit the European geographical dispersion offered by EGI for economic, energetic and / or environmental purposes.

References

1. (2007). US department of energy, voluntary reporting of greenhouse gases: Appendix f. `http://205.254.135.24/oiaf/1605/pdf/Appendix%20F_r071023.pdf`.
2. (2007). US Energy Information Administration (EIA) report. `http://205.254.135.24/electricity/monthly/pdf/chap5.pdf`.
3. (2011). Amazon elastic compute cloud (amazon ec2). `http://aws.amazon.com/fr/ec2/`.
4. (2011). L'augmentation du cout et de la consommation d'energie. `http://www.efficap-energie.com/`.
5. (2011). Stratuslab project. `http://stratuslab.eu/`.
6. Burge, J., Ranganathan, P., and Wiener, J. (2007). Cost-aware scheduling for heterogeneous enterprise machines (cash em). In *Cluster Computing.*, pages 481 –487.
7. Feitelson, D. (2009). Parallel workloads archive. `http://www.cs.huji.ac.il/labs/parallel/workload`.
8. Freeh, V. W., Kappiah, N., Lowenthal, D. K., and Bletsch, T. K. (2008). Just-in-time dynamic voltage scaling: Exploiting inter-node slack to save energy in mpi programs. *J. Parallel Distrib. Comput.*, **68**, 1175–1185.
9. Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.
10. Garg, S., Konugurthi, P., and Buyya, R. (2008). A linear programming driven genetic algorithm for meta-scheduling on utility grids. In *Advanced Computing and Communications,(ADCOM 2008)*, pages 19 –26.
11. Garg, S. K., Yeo, C. S., Anandasivam, A., and Buyya, R. (2011). Environment-conscious scheduling of hpc applications on distributed cloud-oriented data centers. *Journal of Parallel and Distributed Computing*, **71**(6), 732 – 749.
12. Gartner (2007). Gartner estimates ict industry accounts for 2 percent of global CO2 emissions. `http://www.gartner.com/it/page.jsp?id=503867`.
13. Hamilton, J. (2009). Cooperative expendable micro-slice servers (cems): Low cost, low power servers for internet-scale services. In *Proceedings of 4th Biennial Conference on Innovative Date Systems Research (CIDR), Asilomar, California, USA, January*.
14. Hotta, Y., Sato, M., Kimura, H., Matsuoka, S., Boku, T., and Takahashi, D. (2006). Profile-based optimization of power performance by using dynamic voltage scaling on a pc cluster. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, page 8 pp.
15. Kessaci, Y., Melab, N., and Talbi, E.-G. (2011). A pareto-based ga for scheduling hpc applications on distributed cloud infrastructures. In *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, pages 456 –462.
16. Koomey, J. G. (2007). Estimating total power consumption by servers in the U.S. and the world. `http://www.hitecair.com/downloads/cooling_tech.pdf`.
17. Lee, Y. and Zomaya, A. (2010). Energy efficient utilization of resources in cloud computing systems. *The Journal of Supercomputing*, pages 1–13. 10.1007/s11227-010-0421-3.
18. Lee, Y. C. and Zomaya, A. Y. (2009). Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling. In *CC-*

GRID'09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, pages 92–99.
19. Lee, Y. C., Wang, C., Zomaya, A. Y., and Zhou, B. B. (2010). Profit-driven service request scheduling in clouds. In *Cluster, Cloud and Grid Computing (CCGRID)*, pages 15 –24.
20. Lin, M. and Ding, C. (2007). Parallel genetic algorithms for dvs scheduling of distributed embedded systems. In R. Perrott, B. Chapman, J. Subhlok, R. de Mello, and L. Yang, editors, *High Performance Computing and Communications*, volume 4782 of *Lecture Notes in Computer Science*, pages 180–191. Springer Berlin / Heidelberg. 10.1007/978-3-540-75444-222.
21. Mezmaz, M., Melab, N., Kessaci, Y., Lee, Y., Talbi, E.-G., Zomaya, A., and Tuyttens, D. (2011). A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems. *Journal of Parallel and Distributed Computing*, **71**(11), 1497 – 1508.
22. Orgerie, A.-C., Lefevre, L., and Gelas, J.-P. (2008). Save watts in your grid: Green strategies for energy-aware framework in large scale distributed systems. In *Parallel and Distributed Systems, ICPADS '08*, pages 171 –178.
23. Rizvandi, N. B., Taheri, J., Zomaya, A. Y., and Lee, Y. C. (2010). Linear combinations of dvfs-enabled processor frequencies to modify the energy-aware scheduling algorithms. *Cluster Computing and the Grid*, **0**, 388–397.
24. Roy Campbell, Indranil Gupta et al HP Labs, I. I. R. K. U. and Yahoo! (2009). Open cirrus tm cloud computing testbed: Federated data centers for open source systems and services research.
25. S. Greenberg, E. Mills, B. Tschudi, P. Rumsey, B. Myatt. (2006). Best practices for data centers: results from benchmarking 22 data centers. In *Proceedings of the 2006 ACEEE Summer Study on Energy Efficiency in Buildings, Pacific Grove, USA*.
26. Springer, R., Lowenthal, D. K., Rountree, B., and Freeh, V. W. (2006). Minimizing execution time in mpi programs on an energy-constrained, power-scalable cluster. In *Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPoPP '06, pages 230–238, New York, NY, USA. ACM.
27. Talbi, E.-G. (2009). *Metaheuristics: From Design to Implementation*. Wiley Publishing.
28. Tesauro, G., Das, R., Chan, H., Kephart, J. O., Levine, D., III, F. L. R., and Lefurgy, C. (2007). Managing power consumption and performance of computing systems using reinforcement learning. In *(NIPS 2007)*.
29. Venugopal, S., Chu, X., and Buyya, R. (2008). A negotiation mechanism for advance resource reservations using the alternate offers protocol. In *Quality of Service, IWQoS 2008.*, pages 40 –49.
30. Yu, J. and Buyya, R. (2006). Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. *Scientific Programming*, **14**(3-4), 217–230.

Table 5 Experimental comparison for the LLNL Thunder instance, between the MO-GA meta-scheduler algorithm the heuristic and a random approach using an energy oriented selection vector according to the different application arrival rates.

MO-GA vector setting: Energy					
	Value for each criterion				
Arrival rate	Energy (kW h)	CO_2 (Kg)	Profit ($)	Failed applications	Time (sec)
Low	1835115	743149.5	4728480	1094	69054.5
Medium	1955660	871205.5	4705445	1563	18001
High	2622765	1262030	4636565	2406.5	1641
Very high	3076485	1380045	4582340	4157.5	149.5

Used method: Heuristic					
	Value for each criterion				
Arrival rate	Energy (kW h)	CO_2 (Kg)	Profit ($)	Failed applications	Time (sec)
Low	3382620	1530595	4592730	1221.5	151
Medium	3168185	1431930	4588880	1937.5	18.5
High	3206045	1461450	4561565	3130	10
Very high	3298050	1431400	4545470	3493.5	10.5

Used method: Random					
	Value for each criterion				
Arrival rate	Energy (kW h)	CO_2 (Kg)	Profit ($)	Failed applications	Time (sec)
Low	1329660	593546.5	1380050	32303	76
Medium	371989	166044	388355	103123.5	12
High	3591.8	1737.1	3111.1	119828	2
Very high	0	0	0	119849	1

Table 6 Experimental comparison for the RICC instance, between the MO-GA meta-scheduler algorithm, the heuristic and a random approach using an energy oriented selection vector according to the different application arrival rates.

MO-GA vector setting: Energy					
	Value for each criterion				
Arrival rate	Energy (kW h)	CO_2 (Kg)	Profit ($)	Failed applications	Time (sec)
Low	1623135	791343	3699395	66.5	16448.5
Medium	1683800	839701	3695270	97.5	8166.5
High	2349935	1257490	3649005	178	1989.5
Very high	3184630	1641285	3557795	866.5	454

Used method: Heuristic					
	Value for each criterion				
Arrival rate	Energy (kW h)	CO_2 (Kg)	Profit ($)	Failed applications	Time (sec)
Low	3760020	1774590	3484305	66.5	14
Medium	3431830	1575755	3502690	142.5	6
High	4208395	1941405	3417255	324.5	4
Very high	3567975	1701620	3464445	538.5	4

Used method: Random					
	Value for each criterion				
Arrival rate	Energy (kW h)	CO_2 (Kg)	Profit ($)	Failed applications	Time (sec)
Low	1801615	816405	1485300	24833.5	10
Medium	676372	306027	555748	91594	3
High	19265.7	8523.02	14659.7	115375	1
Very high	0	0	0	115855	1

Table 7 Experimental comparison for the LLNL Thunder instance, between the MO-GA meta-scheduler algorithm, the heuristic and a random approach using a profit oriented selection vector according to the different application arrival rates.

MO-GA vector setting: Profit					
	Value for each criterion				
Arrival rate	Energy (kW h)	CO_2 (Kg)	Profit ($)	Failed applications	Time (sec)
Low	2081360	1184045	4805700	1110	70790
Medium	2151975	1207345	4765145	1630	20638.5
High	2795550	1411590	4639730	2817.5	1158
Very high	3099570	1457270	4579790	4437.5	170.5

Used method: Heuristic					
	Value for each criterion				
Arrival rate	Energy (kW h)	CO_2 (Kg)	Profit ($)	Failed applications	Time (sec)
Low	3382620	1530595	4592730	1221.5	151
Medium	3168185	1431930	4588880	1937.5	18.5
High	3206045	1461450	4561565	3130	10
Very high	3298050	1431400	4545470	3493.5	10.5

Used method: Random					
	Value for each criterion				
Arrival rate	Energy (kW h)	CO_2 (Kg)	Profit ($)	Failed applications	Time (sec)
Low	1329660	593546.5	1380050	32303	76
Medium	371989	166044	388355	103123.5	12
High	3591.8	1737.1	3111.1	119828	2
Very high	0	0	0	119849	1

Table 8 Experimental comparison for the RICC instance, between the MO-GA meta-scheduler algorithm, the heuristic and a random approach using a profit oriented selection vector according to the different application arrival rates.

MO-GA vector setting: Profit					
	Value for each criterion				
Arrival rate	Energy (kW h)	CO_2 (Kg)	Profit ($)	Failed applications	Time (sec)
Low	1749095	985186.5	3729270	109	16784
Medium	1854975	1040910	3714575	131.5	7792.5
High	2630490	1456940	3625195	567	1643
Very high	3248800	1648355	3544185	695	486.5

Used method: Heuristic					
	Value for each criterion				
Arrival rate	Energy (kW h)	CO_2 (Kg)	Profit ($)	Failed applications	Time (sec)
Low	3760020	1774590	3484305	66.5	14
Medium	3431830	1575755	3502690	142.5	6
High	4208395	1941405	3417255	324.5	4
Very high	3567975	1701620	3464445	538.5	4

Used method: Random					
	Value for each criterion				
Arrival rate	Energy (kW h)	CO_2 (Kg)	Profit ($)	Failed applications	Time (sec)
Low	1801615	816405	1485300	24833.5	10
Medium	676372	306027	555748	91594	3
High	19265.7	8523.02	14659.7	115375	1
Very high	0	0	0	115855	1

Table 9 Experimental comparison for the LLNL Thunder instance, between the MO-GA meta-scheduler algorithm, the heuristic and a random approach using a CO_2 oriented selection vector according to the different application arrival rates.

MO-GA vector setting: CO_2					
	Value for each criterion				
Arrival rate	Energy (kW h)	CO_2 (Kg)	Profit ($)	Failed applications	Time (sec)
Low	2367775	710355.5	4632040	1093.5	64589.5
Medium	2261425	860452.5	4657870	1617	17150
High	2832295	1287525	4604370	2764	1497
Very high	3205265	1483115	4580650	4225.5	153.5

Used method: Heuristic					
	Value for each criterion				
Arrival rate	Energy (kW h)	CO_2 (Kg)	Profit ($)	Failed applications	Time (sec)
Low	3382620	1530595	4592730	1221.5	151
Medium	3168185	1431930	4588880	1937.5	18.5
High	3206045	1461450	4561565	3130	10
Very high	3298050	1431400	4545470	3493.5	10.5

Used method: Random					
	Value for each criterion				
Arrival rate	Energy (kW h)	CO_2 (Kg)	Profit ($)	Failed applications	Time (sec)
Low	1329660	593546.5	1380050	32303	76
Medium	371989	166044	388355	103123.5	12
High	3591.8	1737.1	3111.1	119828	2
Very high	0	0	0	119849	1

Table 10 Experimental comparison for the RICC instance, between the MO-GA meta-scheduler algorithm, the heuristic and a random approach using a CO_2 oriented selection vector according to the different application arrival rates.

MO-GA vector setting: CO_2					
	Value for each criterion				
Arrival rate	Energy (kW h)	CO_2 (Kg)	Profit ($)	Failed applications	Time (sec)
Low	2617685	766625.5	3527960	65.5	15575
Medium	2343105	799365.5	3587090	105	7462.5
High	2723655	1291920	3582755	387.5	2147.5
Very high	3303150	1806400	3551275	544	203

Used method: Heuristic					
	Value for each criterion				
Arrival rate	Energy (kW h)	CO_2 (Kg)	Profit ($)	Failed applications	Time (sec)
Low	3760020	1774590	3484305	66.5	14
Medium	3431830	1575755	3502690	142.5	6
High	4208395	1941405	3417255	324.5	4
Very high	3567975	1701620	3464445	538.5	4

Used method: Random					
	Value for each criterion				
Arrival rate	Energy (kW h)	CO_2 (Kg)	Profit ($)	Failed applications	Time (sec)
Low	1801615	816405	1485300	24833.5	10
Medium	676372	306027	555748	91594	3
High	19265.7	8523.02	14659.7	115375	1
Very high	0	0	0	115855	1

Table 11 Experimental comparison for the LLNL Thunder instance, between the MO-GA meta-scheduler algorithm, the heuristic and a random approach using an average orientation of the selection vector according to the different application arrival rates.

MO-GA vector setting: Average					
	Value for each criterion				
Arrival rate	Energy (kW h)	CO_2 (Kg)	Profit ($)	Failed applications	Time (sec)
Low	1839645	744802.5	4728630	1094	66760
Medium	1975060	868983.5	4704360	1620.5	17234
High	2661580	1269990	4623960	2405.5	1347.5
Very high	3175135	1450690	4566185	4441.5	168

Used method: Heuristic					
	Value for each criterion				
Arrival rate	Energy (kW h)	CO_2 (Kg)	Profit ($)	Failed applications	Time (sec)
Low	3382620	1530595	4592730	1221.5	151
Medium	3168185	1431930	4588880	1937.5	18.5
High	3206045	1461450	4561565	3130	10
Very high	3298050	1431400	4545470	3493.5	10.5

Used method: Random					
	Value for each criterion				
Arrival rate	Energy (kW h)	CO_2 (Kg)	Profit ($)	Failed applications	Time (sec)
Low	1329660	593546.5	1380050	32303	76
Medium	371989	166044	388355	103123.5	12
High	3591.8	1737.1	3111.1	119828	2
Very high	0	0	0	119849	1

Table 12 Experimental comparison for the RICC instance, between the MO-GA meta-scheduler algorithm, the heuristic and a random approach using an average orientation of the selection vector according to the different application arrival rates.

MO-GA vector setting: Average					
	Value for each criterion				
Arrival rate	Energy (kW h)	CO_2 (Kg)	Profit ($)	Failed applications	Time (sec)
Low	1685740	823813.5	3700325	68.5	15692
Medium	1786950	888938	3691820	96.5	7871
High	2734555	1402205	3589570	330	1715.5
Very high	4349655	2033630	3393760	610.5	380

Used method: Heuristic					
	Value for each criterion				
Arrival rate	Energy (kW h)	CO_2 (Kg)	Profit ($)	Failed applications	Time (sec)
Low	3760020	1774590	3484305	66.5	14
Medium	3431830	1575755	3502690	142.5	6
High	4208395	1941405	3417255	324.5	4
Very high	3567975	1701620	3464445	538.5	4

Used method: Random					
	Value for each criterion				
Arrival rate	Energy (kW h)	CO_2 (Kg)	Profit ($)	Failed applications	Time (sec)
Low	1801615	816405	1485300	24833.5	10
Medium	676372	306027	555748	91594	3
High	19265.7	8523.02	14659.7	115375	1
Very high	0	0	0	115855	1

Table 13 Comparison of the number of failed applications on the LLNL Thunder instance between the MO-GA meta-scheduler algorithm (four different settings of the selection vector), the heuristic and the random approach according to the different application arrival rates.

	MO-GA vector settings				Used method	
Arrival rate	Energy	Profit	CO_2	Average	Heuristic	Random
Low	1094	1110	1093.5	1094	1221.5	32303
Medium	1563	1630	1617	1620.5	1937.5	103123.5
High	1641	2817.5	2764	2405.5	3130	119828
Very high	4157.5	4437.5	4225.5	4441.5	3493.5	119849
Nb applications	119849					

Table 14 Comparison of the number of failed applications on the RICC instance between the MO-GA meta-scheduler algorithm (four different settings of the selection vector), the heuristic and the random approach according to the different application arrival rates.

	MO-GA vector settings				Used method	
Arrival rate	Energy	Profit	CO_2	Average	Heuristic	Random
Low	66.5	109	65.5	68.5	66.5	24833.5
Medium	97.5	131.5	105	96.5	142.5	91594
High	178	567	387.5	330	324.5	115375
Very high	866.5	695	544	610.5	538.5	115855
Nb applications	115855					

Table 15 Improvement rates on the LLNL Thunder instance between the MO-GA meta-scheduler algorithm using an average orientation vector and the heuristic, according to the different application arrival rates.

	Improvement according to criterion (MO-GA meta-scheduler vs heuristic)			
Arrival rate	Energy (Minimization)	CO_2 (Minimization)	Profit (Maximization)	Failed applications (Minimization)
Low	-45%	-51%	+2.9%	-10%
Medium	-37%	-39%	+2.5%	-16%
High	-16%	-13%	+1.3%	-23%
Very high	-3%	+1%	+0.4%	+27%

Table 16 Improvement rates on the RICC instance between the MO-GA meta-scheduler algorithm using an average orientation vector and the heuristic, according to the different application arrival rates.

	Improvement according to criterion (MO-GA meta-scheduler vs heuristic)			
Arrival rate	Energy (Minimization)	CO_2 (Minimization)	Profit (Maximization)	Failed applications (Minimization)
Low	-55%	-53%	+6%	+3%
Medium	-48%	-43%	+5%	-32%
High	-35%	-27%	+5%	+1%
Very high	+22%	+19%	-2%	+13%