# Multi-criteria Scheduling on Clouds

Yacine Kessaci


Ecole Doctorale Sciences Pour l'Ingénieur Université Lille 1 Nord-de-France
Laboratoire d'Informatique Fondamentale de Lille (UMR CNRS 8022)
Centre de Recherche Inria Lille Nord Europe

---

Thèse présentée pour obtenir le grade de docteur

**Discipline : Informatique**

N° d'ordre : 41245

---

# Ordonnancement multi-critère sur Clouds

---

Par : **Yacine Kessaci**

Soutenue le : 28 novembre 2013

Membres du jury :

| | | |
|---|---|---|
| **Président :** | Lionel Seinturier | - Professeur des Universités, Lille 1 |
| **Rapporteurs :** | Pascal Bouvry | - Professeur des Universités, Luxembourg |
| | Jean-Marc Pierson | - Professeur des Universités, Toulouse 3 |
| **Examinateur :** | Laurent Lefèvre | - Chargé de recherche HDR, ENS Lyon/LIP/Inria |
| **Directeurs de thèse :** | Nouredine Melab | - Professeur des Universités, Lille 1 |
| | El-Ghazali Talbi | - Professeur des Universités, Lille 1 |

# Remerciements

Cette belle aventure qu'est le doctorat m'a appris beaucoup de choses aussi bien sur le plan scientifique que humain. Il va sans dire que ces trois dernières années de réflexion et de travail ont renforcé mes connaissances en plus de m'en faire découvrir de nombreuses nouvelles. Cependant une partie non négligeable de cet apprentissage ne concerne pas la recherche en elle même. En effet, la thèse de doctorat est un projet que j'ai vécu avant tout comme une aventure humaine à travers de belles rencontres qui ont contribué de près ou de loin à l'aboutissement de ce long périple. Il est normal donc de leur adresser mes remerciements.

Je tiens tout d'abord à exprimer l'expression de mon respect et de ma profonde reconnaissance à mes deux directeurs de thèse : les professeurs Nouredine Melab et El-Ghazali Talbi. La pertinence de leurs conseils ainsi que leur lucidité scientifique ont été précieux dans la réalisation et l'aboutissement de ce travail de recherche.

Mes remerciements vont également à tous les membres du jury qui m'ont fait l'honneur d'évaluer mon travail pour leur disponibilité, les conseils et remarques constructives émis dans leur rapport. Je pense ici à Lionel Seinturier, Pascal Bouvry, Jean-Marc Pierson et Laurent Lefèvre.

Ces remerciements seraient incomplets si je n'en adressais pas à l'ensemble des membres de l'équipe DOLPHIN pour leur accueil ainsi que pour la très bonne ambiance que j'ai trouvée au centre durant ces trois années de thèse. Un merci en particulier à Ahcène, Aline, Benjamin, Ekatarina, François, Julie H, Julie J, Karima, Khedidja, Marie-Eléonore, Martin, Mathieu, Mostepha, Moustapha, Nadia, Sophie, Thé Van et Tuan. Sans oublier les permanents de l'équipe : Arnaud, Bilel, Clarisse, Clive, Dimo, François, Laetitia et Luce. Ni les anciens membres de l'équipe : Mohand Mezmaz, Lekhdar Loukil et Malika Mehdi.

Je voudrais aussi adresser mes remerciements à mes amis d'ici ou d'Algérie.

Et enfin, le plus important, mes chaleureux remerciements vont à ma famille, avec une pensée particulièrement affectueuse pour mes parents et mon frère Wanis pour leur confiance et leur soutien indéfectible. Qu'ils sachent que dans mon cœur ce travail est autant mien que le leur.

# Contents

# Introduction

During the last decade, the cloud computing paradigm has carved out a significant share of the IT field. It appears nowadays to be increasingly adopted in many areas by proposing several services. Thus, the effective access to, and the use of those services raises several issues, and scheduling is one of the major of them. Even if the problem has largely been addressed in the literature, it should be revisited within the context of cloud computing. Indeed, the flexibility and the dynamicity offered by the cloud and some major challenges such as energy reduction increase the scheduling complexity and make difficult to find efficient solutions to those issues in a reasonable time. Therefore, applying exact optimization methods turns out to be illusory because of the computation time. As a solution, multi-objective metaheuristics constitute a good alternative. However, these metaheuristics as well as the modeling of the problem need to be rethought to take into account both the specifications and the constraints of the problem of scheduling on clouds.

The model of cloud computing is a factorization of the resources for simplification purposes. It offers an easy access to those resources anywhere at any time. The cloud computing is based on both IT and economic concepts [Armbrust 2009, Buyya 2008]. The IT concept concerns the infrastructure used for providing the resources, it is derived mainly from the grid computing model enhanced by a flexibility offered by different software frameworks. On the other hand, the economic concept is in charge of the business interaction of the cloud paradigm with the clients. One can see then through this brief description how the cloud computing inherits from the issues related to both grid and economic concepts. Indeed, the economic concept adds totally new constraints and objectives affecting the scheduling over a new distributed architecture such as cloud. Moreover the flexibility associated to the size variability of a cloud brings scheduling issues mainly related to the different levels at which one can consider the cloud scheduling: service-level scheduling, task-level scheduling and VM-level scheduling.

Therefore, if one considers the highest level of the cloud where the scheduling concerns an abstract distributed infrastructure dealing with coarse grained services [Campbell 2009], one deals with the market-oriented service-level scheduling. At this level, the scheduling problem has its proper constraints and objectives directly related to its context. Indeed, scheduling at this level in addition to consider objectives such as, increasing the profit of the provider or reducing the energy consumption of such huge infrastructures, needs above all a calibration of the optimization techniques by finding the proper efficient drivers to this level of scheduling. The works in [Garg 2010] and in [Wu 2013] deal with this level of scheduling. However, several questions are still open, such as the way how one can better benefit from the characteristic of the infrastructures by using for

instance more detailed models, or how addressing the different conflicting objectives simultaneously can represent an interesting challenge to enhance the obtained results of the faced objectives.

The market-oriented characteristic of the cloud leads naturally to put the satisfaction of the clients as a main optimization target. However, the satisfaction is related to the accuracy of the provided service and depends on several criteria according to the type of cloud dealt with. Therefore, an evolution of the service-level scheduling model with more precise services designed to better fit the clients' needs is required. This level is called the task-level scheduling. Lowering that way the scheduling level adds precisions in the requested services leading generally to host tasks over detailed resources configurations such as in [Ama 2012b] and in [Wu 2013]. Task-level scheduling brings also its proper specifications and constraints that have to be considered in the scheduling model. At such a level with detailed requests, some other new challenges emerge and still need to be addressed such as the tradeoff between the price and the performance regarding the client's satisfaction. Besides, all the aforementioned levels are tightly related to the lowest level from which one can see the cloud computing. Thus, despite the market-orientation of the cloud, the latter remains based on IT concepts inherited from the general grid concepts. This level as the other ones raises challenges and requires an optimization process. The optimization is mainly related to scheduling the smallest cloud entities (Virtual Machines) over the hardware infrastructure such as in [Verma 2008, Xu 2010, Laszewski 2009]. Thus, the main challenge of the VM-level scheduling is to offer the best policy that optimizes the use of the hardware infrastructure regarding some criteria to be as much as possible at the service of the upper market scheduling levels.

Through what has been previously said, the different existing scheduling approaches related to the levels of views that one can have of the cloud show several defects. One can summarize them as, a weakness in the proposed schedulers based generally only on simple heuristics, a lack in the adaptivity of the approaches according to the cloud levels, to the best of our knowledge almost no Pareto multi-objective solving algorithms exist and finally very few realistic experiments facing the real cloud constraints.

Therefore, dealing with those lacks and facing them requires highlighting and addressing both scientific and technical challenges in order to rethink the scheduling optimization over the cloud computing paradigm. In that purpose, the contribution of this thesis faces all those challenges and provides solutions at all the scheduling levels using different metaheuristic algorithms. This thesis offers the bridge between the metaheuristics and the cloud scheduling by proposing an appropriate solution depending on the level of scheduling and/or the business trend of each cloud (see. Figure 1). Therefore, in this document we propose specific metaheuristic-based schedulers that fit the related constraints and the characteristics of each scheduling

level in the cloud. We also propose new and updated optimization models regarding each addressed criterion over all the different cloud levels. Moreover, due to the conflicting objectives related to all the different levels of the cloud, we propose a Pareto approach for each level to tackle simultaneously all the criteria and find the best tradeoff. Finally, because of the Pareto nature of the proposed solutions, we develop a selection mechanism related to both the needs and the features of each scheduling level to propose each time the solution among the proposed Pareto front that best suits the cloud situation. In addition, all the proposed works have been subject to thorough experiments including both artificial and real scenarios. Hence, in order to validate all the different solutions for each cloud scheduling level, we used data centers workload archives [Feitelson 2009], Amazon's data pricing [Ama 2012a] and conducted real cloud deployment over up to 200 machines from the GRID'5000 infrastructure [GRI 2013].

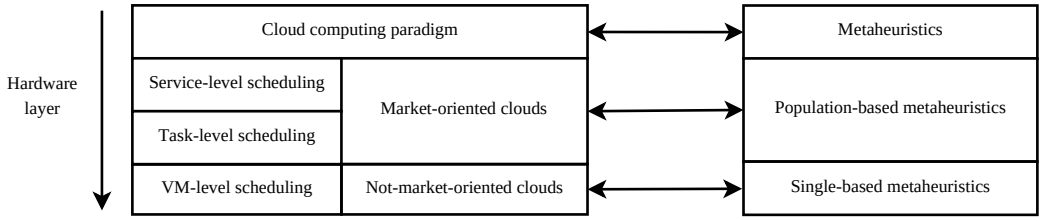



Figure 1: A General overview of the cloud computing scheduling levels and their associated solving metaheurstics.

## Document Organization

The first chapter presents all the background and prerequisites needed to the general comprehension of the manuscript, including the cloud computing concepts and its related optimization techniques. In this purpose, we first introduce the cloud computing paradigm in the context of scheduling in addition to a classification of the different types of clouds and a survey of the cloud managers. Furthermore, a background on both multi-objective combinatorial optimization and the different families of metaheuristics is described. Thereafter, a general formulation of the cloud scheduling problem and the different scheduling levels related to the cloud computing architecture are presented. Finally, we propose a related work and a classification of the different cloud scheduling approaches of the literature.

In the second chapter, we tackle the issues related to service-level scheduling. In that purpose, we consider the relationship between energy, green house gas emissions and profit and pay attention on how each of those criteria can affect the others. We use for that a multi-objective evolutionary algorithm as a meta-scheduler that allows obtaining a Pareto set of solutions and showing the trade-off between all the tackled criteria. Finally, a bunch of realistic experiments are performed on a long period of

real workloads for the comparison of our meta-scheduler to two different approaches.

The third chapter contributes to propose a more accurate model that enhances the quality of the service by tackling the main parameters that affect task-level scheduling on a cloud brokering environment. The two parameters are the prices of the Virtual Machines (VM) instances and their response time when hosting the client's tasks. In that purpose, we propose a multi-objective genetic algorithm for cloud brokering that aims to dynamically minimize both the parameters in order to provide the best quality of service (QoS) to the clients while offering an interesting profit for the cloud broker. A Pareto set gives the best assignments of the client's tasks over the combination of VM instances. The experimental validation of our contribution is based on real information provided by the infrastructure service provider (e.g. Amazon) to retrieve the prices of the instances and their different performances.

In Chapter four, we address the issues related to the lowest and core level of the cloud infrastructure by proposing efficient optimization algorithms for the VM-level scheduling. The main issues related to such an infrastructure are both its energy consumption and provided performance. Thus, the first scheduler that we propose (EMLS-ONC) is based on a multi-start local search metaheuristic that provides a near-optimal or optimal scheduling by dispatching the incoming VMs according to the minimum energy consumption. As a second part, we tackle the VMs performance issues within a cloud infrastructure using a Pareto bi-objective version of the first scheduler (EMLS-ONC-MO) that addresses simultaneously both the energy consumption criterion and the performance of the VMs. To be as realistic as possible, the experiments that we present to validate our algorithms are done on both artificial and real cloud infrastructures deployed with the OpenNebula cloud manager.

In the last part, the conclusion, the perspectives and the appendices of this thesis are presented.

Chapter 1

# Metaheuristics for Cloud Scheduling

**Main publications related to this chapter**

- Y. Kessaci, M. Mezmaz, N. Melab, E-G. Talbi, D. Tuyttens. Parallel Evolutionary Algorithms for Energy Aware Scheduling. *Intelligent Decision Systems in Large-Scale Distributed Environments Series: Studies in Computational Intelligence, Springer Berlin Heidelberg*, 2011, Vol. 362 Bouvry et al. (Eds.).(Chapter4), pp. 75-100.

- M. Mezmaz, N. Melab, Y. Kessaci, Y. Lee, E-G. Talbi, A. Zomaya, D. Tuyttens. A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems. *Journal of Parallel and Distributed Computing*, 2011.

**Contents**



## 1.1 Introduction

The cloud computing paradigm is depicting more and more the field of IT. The model of the cloud computing is derived from the grid computing model, extended with economic concepts [Foster 2008]. Therefore, cloud computing inherits from well known issues of both aforementioned disciplines. Among these issues, one

can consider the scheduling as being a major one. This problem is combinatorial and has been proven to be NP-hard [Garey 1979]. It is also often multi-objective since several criteria brought from economics for instance may be considered. Using combinatorial optimization techniques such as metaheuristics appears to be a good alternative to deal with scheduling.

Therefore, this first chapter presents all the background and prerequisites, related to the cloud computing and its associated scheduling issues, needed to the general comprehension of the manuscript.

First, we describe the cloud computing paradigm in the context of optimization. In addition, we present a classification of the different types of clouds and a general survey of the existing cloud managers. Thereafter, a background on combinatorial optimization, an overview of the principles and the different families of metaheuristics and a presentation of the ParadisEO framework used to tackle and implement our contributions are introduced. Finally, a general formulation of the cloud scheduling problem and a literature-review of the different cloud scheduling approaches are proposed.

## 1.2 Background on Cloud Computing

Cloud computing is an emerging paradigm hardly describable by a unique definition. It introduces simplicity and flexibility and also helps to provide different services in a ubiquitous way. However, these features are highly dependent on the quality of the cloud scheduler.

In this section, we present the different taxonomies and classifications related to the topic we address, cloud computing. First, we introduce the cloud terminology and its definition. In the second part we present the related taxonomy of cloud computing according to the provided services. In a third part we classify the different types of scheduling in the cloud computing, and finally we elaborate a survey for the different cloud managers with a focus on OpenNebula.

### 1.2.1 Cloud computing paradigm

#### 1.2.1.1 Terminology

For a better understanding of the manuscript, we present in the following the common terminology of the cloud scheduling.

- **Virtual Machine (VM):** is a software based machine emulation technique that provides a desirable and on demand computing environment to users.
- **Virtualization:** it is the VMs related concept. Virtualization consists to run 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.

- **Consolidation:** the task consolidation is an effective method to increase resource utilization and to reduce energy consumption. The task consolidation problem 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.

- **Hypervisor:** also called virtual machine monitor (VMM) is a piece of computer software that manages a hardware infrastructure in order to create and run seamlessly several virtual machines on top of it.

- **Data center:** is another way to define a private cloud dedicated to private use (see Section 1.2.1.3).

- **Host:** represents the hardware entities that compose a data center.

- **Provider:** in its general sense it represents an entity charged to offer the clients their requested services in exchange or not of a remuneration. When the owner of the cloud is not unique we talk about multi-providers. This can happen for instance with a federation of clouds called a multi-cloud.

- **Broker:** it is a third-party with the role of finding in exchange of remuneration the provider that provides the service that best fits the user's needs. The action done by the broker is called the brokering.

- **Client:** the client asks the provider or the broker for services in order to satisfy his/her needs. When the service is not charged we can simply call him/her a user.

- **Service:** it represents a commodity which consists of the provision of a technical support varying from a cloud to another (see Section 1.2.1.3) to satisfy the client's needs. The service can be charged or not.

- **Task:** also called application, the task represents a software entity sent by the user to achieve his/her objectives. It has some requirements that have to be satisfied by the provider to meet the Quality of the Service (QoS) (see Section 1.2.1.2).

#### 1.2.1.2 Cloud computing definition

Nowadays, the naive concept of cloud computing refers to the Internet. It consists in the migration of the personnel computers and utilities on the network, more precisely on the Internet. This definition is made of the observation that the common people have in mind of the cloud computing. However, different definitions have been formulated in the literature.

*Armbrust et al.* [Armbrust 2009] note that "*Cloud computing refers to both the applications delivered as services over the Internet and the hardware and system software in the data centers that provide those services*". This definition concerns

the last evolution of clouds. Indeed, moving the hardware infrastructure in this definition from private computers includes also the movement of the software services to data centers accessed via Internet. Another definition of the cloud is proposed by *Buyya et al.* in [Buyya 2008], it presents the cloud as "*a type of parallel and distributed system consisting of a collection of interconnected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements*". This definition introduces the Service Level Agreements (SLAs) in the cloud. This adds to this new paradigm a market-oriented side and highlights its economic nature. The definition that we introduce is a mixed definition that fits the different types of cloud that we dealt with through all the works done in this thesis. We define the cloud as "*An evolution of parallel and distributed systems, including dynamicity, transparency and flexibility, interconnecting virtualized computers offering market-oriented or non-market-oriented services*".

The main aspects that emerge from all the above definitions is that the cloud allows to provide services, from the basic one with a simple infrastructure to the complex one with a complete software package. It is also an evolution of the previous parallel and distributed infrastructures like grids and can offer the previously cited services with or without a market-oriented policy. In the case of a market-oriented cloud, the service consumption follows *pay-as-you-go* basis.

In our definition we introduced the word *evolution*, this refers to the upgrades offered by the cloud computing compared to its infrastructure ancestors. Thus, the previous parallel and distributed infrastructures targeted narrow objectives for either the class of users or the type of provided utilities. Cloud computing allows to cover a much wider audience helping to democratize the remote access to the resources. The services the cloud provides stretch from an end-user who wants to store a personal document or use a remote web application, to a huge company outsourcing its whole IT infrastructure to third-party data centers provider. These evolutions were made possible by technical and economical transformations. The technical ones concern the capacity of the cloud to take advantage from the improvements brought by the virtualization technology [Vir 2013, Barham 2003]. Those advantages such as flexibility, transparency and reliability helped to open new horizons.

For instance, scheduling makes easier the way to tackle different IT issues like energy-efficiency while holding a satisfactory level of service. The satisfactory level of service gave birth to a marketization of the cloud with as mentioned before concepts like SLAs which define the Quality of Service (QoS) set between the clients and the cloud providers. The SLA represents a contract between the costumer and the provider. It specifies the details of the agreement about the service based on metrics, and establish penalties in case of contract violating. The SLA allows one to reassure the customers in their idea to move their business to the cloud by providing a warranty. As said previously, moving to the cloud has benefited both the companies and end-users. The former, saves extra resources and maintenance by getting the exact needed resources at each time. Similarly, the latter in addition to get a continuous access to their data, they can use software on demand without buying

it. Lately, the development of the different types of services and providers has complicated the user's choice. Thereby, a new third-party, the broker, emerged between the client and the provider.

#### 1.2.1.3 Cloud computing classification

Cloud computing can be classified hierarchically following two branches. The first branch is related to the architecture of the cloud, while the second is more related to the type of service provided by the latter.

All cloud infrastructures offer the same characteristic no matter the service type that they propose. They give the possibility to the user to obtain and dispose of the resources on demand and the ability to get access to them from anywhere in the world. This provides transparency that is reflected to the user by a single gate as an access point whatever the amount of proposed resources behind this gate is. However, depending on how those resources are managed, clouds fall into three categories [Rimal 2009]:

- *Private Cloud:* in this type of clouds, all the components are owned and managed by the organization. The cloud is not subject to any restriction or bounding. The data and the processes running on the cloud are not bounded regarding any resources or exposed to any security of legal requirements.
- *Public Cloud:* this most common type represents the cloud in its general market sense, where the resources are provided over the Internet in a transparent way, dynamically and on-demand according to the users' needs. Because of the shared aspect of the infrastructure between different users, the provided services are exposed to more restrictions. These are whether physical depending on the availability of the resources according to the customer request, or legal regarding the security and the confidentiality of the exchanged or hosted data and information.
- *Hybrid Cloud:* this last type combines both previously presented types. It is composed of multiple internal and/or external providers with different management policies. This type of cloud can typically be used by a company that has its private cloud, but because of higher needs, subscribes to another provider for extra needs. Therefore, the processes hosted on its private cloud will not be exposed to the same rules as the ones hosted in the subscribed cloud.

Depending on the addressed problem, in the following chapters we deal whether with a public cloud or a private one.

In each of the cloud types presented above, one can find different categories of proposed services (Table 1.1 [Buyya 2009]). The general notation representing those categories is XaaS where X is a service such as software, platform, infrastructure etc. In the following, we discuss the most important provided services in the cloud infrastructures.

- *Software as a Service (SaaS):* the Software as a Service cloud is based on the concept of flatshare. A single software is shared by several costumers. The provider relies on a duplication of the software by proposing for each user an instance of the code and of the database layer. This allows the provider to satisfy several requests simultaneously. The provider proposes applications as services, SaaS can be therefore affiliated to the Application Service Provider (ASP) model. The development of the software sharing over the SaaS clouds is conducting the software companies to release more often their software on this type of architecture. An example of the leading SaaS clouds is presented in Table 1.1.

- *Platform as a Service (PaaS):* this type of cloud is more oriented to serve the development companies. It offers preconfigured environments, to facilitate and minimize the development effort. Indeed, compared to conventional application development, using the PaaS offers a huge scalability, adaptivity and flexibility that slash the preconfiguring development time. The leader providers of those types of clouds are Google AppEngine, Microsoft Azure, etc.

- *Hardware as a Service (HaaS):* this model of cloud is dedicated more to the business users helping the startup companies or the ones that do not want to invest in data centers to fit their hardware needs. According to Nicolas Carr says, it is [1] *"the idea of buying IT hardware -or even an entire data center- as a pay-as-you-go subscription service that scales up or down to meet your needs. But as a result of rapid advances in hardware virtualization, IT automation, and usage metering and pricing, I think the concept of hardware-as-a-service -let's call it HaaS- may at last be ready for prime time".*

- *Infrastructure as a Service (IaaS):* IaaS is the basis of the other cloud models. The major advantage is that thanks to the offered flexibility, the companies fit to their growth easily in terms of infrastructures, benefiting from the on-demand payment process. The other major advantage is the use of up-to-date hardware with the latest technology. As a result, the customers benefit by saving time and money. There is almost no difference between the HaaS and IaaS models. However, we can say that the IaaS model is the market formalization of the HaaS concept. The leading providers of IaaS in the world are Amazon EC2, GoGrid, Rackspace, etc.

Note that all the proposed works in this manuscript deal with the IaaS cloud.

### 1.2.2 OpenNebula Cloud manager

Lately, a significant evolution in the field of the parallel and distributed computing led to a massive utilization of clouds. Therefore, virtualization and consolidation techniques have been integrated in this new model through middlewares called

[1] http://www.roughtype.com/

Table 1.1: Cloud architecture types

| Category | Characteristics | Product Type | Vendors & Products |
|---|---|---|---|
| SaaS | Customers are provided with applications that are accessible anytime and from anywhere | Web applications and services (Web 2.0) | SalesForce.com (CRM); Clarizen.com (Project Management); Google Documents; Google Mail (Automation) |
| PaaS | Customers are provided with a platform for developing applications hosted in the Cloud | Programming APIs and frameworks; Deployment system | Google AppEngine; Microsoft Azure; Manjrasoft Aneka |
| IaaS/HaaS | Customers are provided with virtualized hardware and storage on top of which they can build their infrastructure | Virtual machines management infrastructure, Storage management | Amazon EC2 and S3; GoGrid; Nirvanix |

cloud managers. We can cite as cloud managers: Openstack [Ope 2012], OpenNebula [Milojicic 2011], Eucalyptus [Nurmi 2009], Nimbus [Keahey 2007], etc. Those middlewares using a hypervisor, help to exploit the data center in a seamless and transparent way by running Virtual Machines (VM). We will discuss in the following the major open source cloud managers with a focus on OpenNebula to know more about how it operates and the different components that compose it. The same analysis can be done over the market-oriented cloud managers like Amazon EC2 [Ama 2012b], GoGrid [GoG 2013], etc. The only difference is that for the latter the source code is not open and follows each provider specifications. The OpenNebula cloud manager features are presented in the following:

- *Computing Architecture:* it is a cluster into an IaaS cloud, focused on the efficient, dynamic and scalable management of VMs within a private cloud involving a large amount of virtual and physical servers.
- *Virtualization Management:* it uses Xen, KVM and on demand access to Amazon EC2.
- *Service:* infrastructure as a Service IaaS.
- *Load balancing:* it is based on Nginx Server configured as a load balancer, based on round-robin or weighted selection mechanism.
- *Fault tolerance:* the deamon can be restarted and all the running VMs recovered, it includes also a persistent database backend to store host and VM information.
- *Interoperability:* as managing private clouds, it offers interoperability between intra cloud services.

- *Storage:* regarding the database, it is a persistent storage for ONE data structures with a SQLite3 backend as a core component of the the internal data structures.
- *Security:* a firewall and a Virtual Private Network Tunnel.
- *Programming Framework:* using Java, Ruby and C++.

Table A.1 proposed in [Rimal 2009] and presented in the Appendix A summarizes the different open source based cloud managers, their different components and characteristics.

## 1.3 Background on Multi-objective Metaheuristics

Through all previously described cloud scheduling issues and the different lacks of the current used algorithms for tackling those issues. We propose in this manuscript multi-objective metaheuristics to address all the aforementioned misses. To better understand the metaheuristic concepts, this defines the concept of multi-objective optimization and metaheuristics with a focus on their different features.

### 1.3.1 Multi-objective combinatorial optimization

An optimization problem consists in an optimization (minimization or maximization) of a unique cost function in case of a mono-objective optimization or a vector of cost functions when dealing with a multi-objective optimization. The cost function is formally called objective function (see Section 1.3.2.1).

A multi-objective optimization problem is defined as:

$$(MOP) = \begin{cases} Min\ F(x) = (f_1(x), ..., f_{nb_{obj}}(x)) \\ s.t.\ x \in S \end{cases} \tag{1.1}$$

where $nb_{obj}$ ($nb_{obj} \geq 2$) is the number of objectives and S represents the set of feasible solutions associated with equality and inequality constraints and explicit bounds.

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 a 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 (see Figure 1.1). 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 [Pareto 1896].

The definition of a mono-objective optimization problem is equivalent to the definition of a multi-objective one, where the number of objective functions equals 1 and the decision space is uni-dimensional.

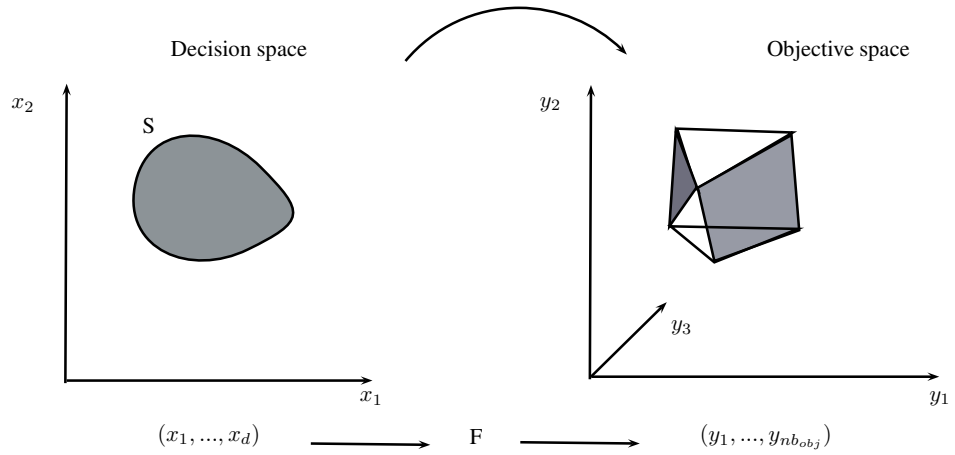



Figure 1.1: Decision space and objective space in a MOP.

As shown in Figure 1.2, the resolution methods of a problem are various and numerous. Using one or another depends mainly on the complexity of the addressed problem. Those methods belong to two main families: exact methods and heuristics. Exact methods (e.g. branch-and-x, dynamic programming or constraints programming) allow to find the best solutions and guarantee their optimality. However, they are time consuming and become quickly impractical for large problems. Conversely, heuristics produce near optimal (sometimes optimal) solutions in a reasonable time. They are quite practical for large-size problem instances. The heuristics family splits into two types. The first type of heuristics is specific to the problem and usually designed exclusively to solve a particular problem and/or instance. The second type is more generic and allows one to address different problems. In this case, they are called metaheuristics. These latter are based on the iterative improvement of either a single solution (e.g. hill climbing, simulated annealing or tabu search) or a population of solutions (e.g. evolutionary algorithms or ant colonies) of a given optimization problem. In this document, the focus will be exclusively on metaheuristics.

#### 1.3.1.1 Concepts and definitions

For minimization problems, the Pareto concepts of MOPs are defined as follows (for maximization problems the definitions are similar):

$$\begin{cases} \forall i \in [1..nb_{obj}], & y_i^1 \leq y_i^2 \\ \exists j \in [1..nb_{obj}], & y_j^1 < y_j^2. \end{cases}$$

- *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 $y^1$,

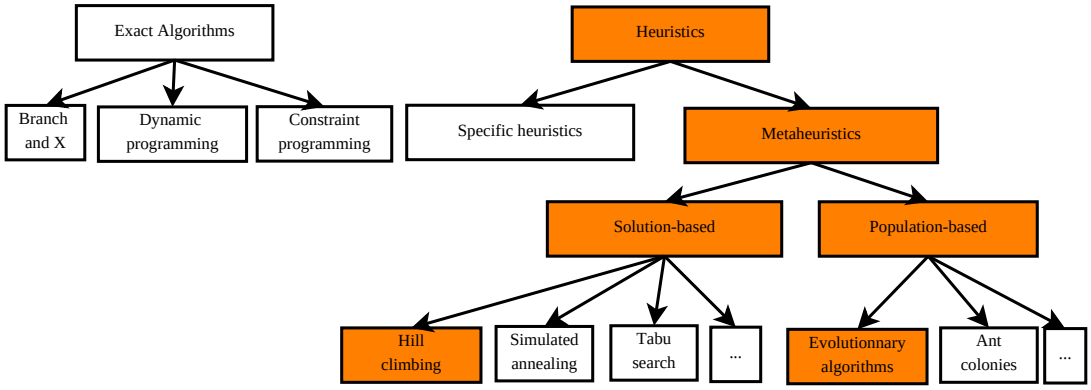



Figure 1.2: Taxonomy of different resolution approaches for optimization problems.

and at least one component of $y^2$ is greater than its correspondent in $y^1$. The generally used concept is Pareto optimality. Pareto optimality definition comes directly from the dominance concept. The concept was proposed initially by *F.Y. Edgeworth* in 1881 [Edgeworth 1881] and extended by *V. Pareto* in 1896 [Pareto 1896]. A Pareto optimal solution denotes that it is not possible to find a solution that improves the performances on a criterion without decreasing the quality of at least another criterion.

- *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 $F(x)$. 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 Figure 1.3).

- *Pareto optimal set*: for a MOP (F, S), the Pareto optimal set is the set of Pareto optimal solutions. It is defined as $P^* = x \in S / \nexists\ x' \in S, F(x') \prec F(x)$.

- *Pareto front*: for a MOP, the Pareto front is the image of the Pareto optimal set in the objective space. It is defined as $PF^* = F(x), x \in P^*$.

#### 1.3.1.2 Multi-objective resolution approaches

Scheduling in cloud infrastructures can be mono-objective or multi-objective. The mono-objective approaches, such as [Garg 2008, Burge 2007, Zhuo 2008, Feller 2012], aim to optimize only one criterion under some constraints. While multi-objective ones deal with several conflicting criteria. The goal of a multi-objective optimization method is to find a good compromise between these objectives.

As shown in Figure 1.4, it is possible to identify three main categories according to the selected multi-objective approach: Aggregation, lexicographic and Pareto approaches.

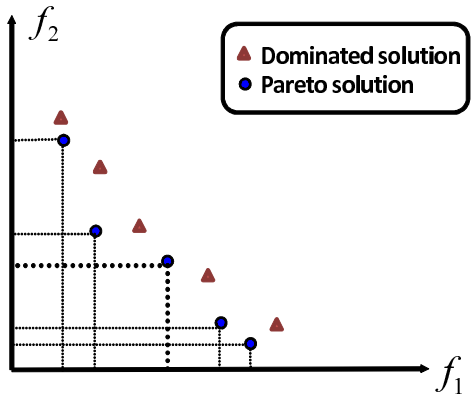



Figure 1.3: Non-dominated solutions in the objective space.

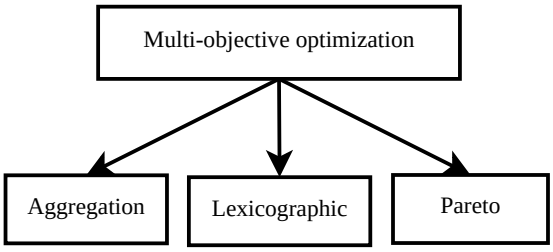



Figure 1.4: Classification according to the optimization approaches.

**Aggregation approach** The aggregation (or weighted) method is one of the first and most used methods for modeling a multi-objective problem. It consists in using an aggregation function to transform a multi-objective problem into a mono-objective problem by combining the various objective functions into a single objective function generally in a linear way. The obtained results in the resolution of the problem depend strongly on the parameters chosen for the weight vector.

As an example, the work [Lee 2009], address the task scheduling problem on heterogeneous computing systems (HCSs) and propose an energy-conscious scheduling heuristic (ECS) that takes into account the completion time and energy consumption. It balances these two performance goals using a novel objective function called relative superiority (RS).

Another example is in Equation (3.1), where the satisfaction parameter is an aggregation of two primary objectives the service response time and its cost.

**Lexicographic approach** In this traditional approach, the search is carried out according to a given preference order of the objectives. This order defines the significance level of the objectives. Therefrom, a set of mono-objective problems are solved in a sequential manner. If the problem associated with the most significant objective function has a unique solution, the search provides the optimal solution and stops. Otherwise, the problem associated with the second most significant objective function is solved. The same stopping criteria and process are iterated

until the treatment of the last function.

As an example of lexicographic approach we can find the works in [Rizvandi 2010, Garg 2010]. In [Rizvandi 2010] the authors investigate the task scheduling problem using a heuristic based on two stages called Maximum Minimum Frequency DVFS (MMF-DVFS). The goal of the first stage is to find a schedule of tasks that minimizes the makespan. The second stage tries to find the right setting of the processor to minimize energy consumption without changing the makespan of the schedule provided by the first stage. In [Garg 2010] different heuristics are proposed. The difference between them is the order of prioritization between the objectives.

**Pareto approach** The Pareto approach uses the concept of dominance in the fitness assignment, contrary to the other approaches that use a scalarization function or treat the various objectives separately. The main advantage of the Pareto approach is it does not need the transformation of the multi-objective problem into a mono-objective problem. In a single run, it is able to generate a diverse set of Pareto solutions in the concave portions of the convex hull of feasible objective space.

For example, [Xu 2010] and [Khan 2009] are examples of methods that use a Pareto approach. In [Xu 2010], the authors propose a multi-objective fuzzy genetic algorithm to simultaneously minimize the total resource wastage, the power consumption and the thermal dissipation cost. In this algorithm, the best solution is the one that belongs the most to each fuzzy set of each objective. In [Khan 2009], a cooperative game model based on the concept of Nash Bargaining Solution (NBS) is proposed. NBS allows guaranteeing the Pareto optimality of the approach. The aim of this work is to simultaneously minimize the energy consumption and the makespan subject to the constraints of deadlines and tasks' architectural requirements.

### 1.3.2 Metaheuristics

Unlike exact methods, metaheuristics allow to tackle large-size problem instances by delivering satisfactory solutions in a reasonable time [Talbi 2009]. There is no guarantee to find global optimal solutions or even bounded solutions. Metaheuristics have received more and more popularity in the 20 past years. Their use in many applications shows their efficiency and effectiveness to solve large and complex problems. Application of metaheuristics falls into two main branches. The first branch includes the single-solution based metaheuristics (S-metaheuristics) and the second the population-based metaheuristics (P-metaheuristics).

Those two families of metaheuristics share several principles but are complementary in their behaviors. Indeed, while the S-metaheuristics have more a local search based on iterations that help to transform and improve one solution by intensifying the research around its neighborhood, the P-metaheuristics explore a bigger search space by involving the whole set of individuals (population). The individuals evolve together for more diversification in the search space toward better solutions. In the next chapters of this document, we will use methods from both families.

#### 1.3.2.1 Modeling

**Solution representation** Designing any metaheuristic needs an encoding (representation) of a solution[2]. It is a fundamental design question in the development of metaheuristics. The encoding plays a major role in the efficiency and effectiveness of any metaheuristic. In addition, the encoding must be suitable and relevant to the tackled optimization problem. Moreover, the efficiency of a representation is also related to the search operators applied on this representation (neighborhood, recombination, etc.). One can classify the combinatorial optimization encodings according to the tackled problem into four representations (see Figure 1.5): vector of real values (continuous optimization, parameter identification, etc.), binary encoding (knapsack problem, SAT problem, etc.), permutation encoding (sequencing problems, traveling salesman problem, scheduling problems, etc.) and finally, vector of discrete values (location problem, assignment problem, etc). The vector of discrete values is also commonly used to model scheduling on cloud architectures.

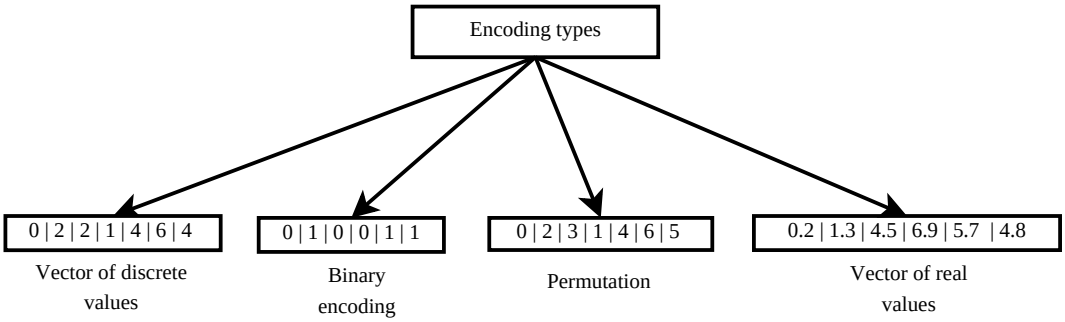



Figure 1.5: Main encodings for optimization problems.

**Objective function** The objective function $f : S \longrightarrow \mathbb{R}$ formulates the goal to be achieved. It associates to each solution of the search space $S$ an absolute value that gives the quality or the *fitness* of the solution allowing a complete ordering of all solutions of this search space. The objective function is an essential element in designing a metaheuristic as it guides the search toward "good" solutions of the search space. If the objective function is improperly defined, it can lead to non-acceptable solutions whatever is the used metaheuristic.

#### 1.3.2.2 Solution methods

**S-Metaheuristics** Single-based metaheuristics (S-metaheuristics) are iterative methods commonly used in the optimization field. They proved their efficiency in solving real complex problems. As said in [Crainic 2003], they can be viewed as "walks" through neighborhoods or search trajectories through the search space of

[2]In the evolutionary computation community, the genotype defines the representation of a solution. A solution is defined as the phenotype.

the problem at hand. The walks (or trajectories) are performed by iterative procedures that move from the current solution to another one in the search space. The principle of a S-metaheuristic is given in Algorithm 1. The algorithm starts by an initial solution that is usually randomly generated. This initial solution represents the first current solution. At each iteration of the algorithm, the current solution is replaced by another solution from its neighborhood. One or several evaluation function(s) assign(s) to each solution a value called the *fitness*. This fitness gives to the algorithm the ability to identify the interesting solution for selecting it as a current solution of the next iteration. The selection policy differs from an algorithm to another (e.g. best improvement, first improvement, random selection, etc.). Besides, there are several techniques to generate the neighborhood from a current solution. One uses for that a *neighborhood operator*. According to the desired neighborhood, neighborhood operators can use the exchange process, the insertion process, the swap process, etc.

In this document we use a type of S-metaheuristics called hill climbing (see Algorithm 2). The hill climbing algorithm inherits from the major aforementioned concepts. It starts with a randomly generated initial solution. At each iteration, the heuristic replaces the current solution by a neighbor that improves the objective function. The algorithm stops when no more solution in the neighbor of the current solution is better than the current solution itself. This means that the algorithm reached the local optimum.

**Algorithm 1** The general template of a S-metaheuristic algorithm

1: Initial Solution ($s_0$)
2: $t := 0$;
3: **repeat**
4: $s'(t)$:= ApplyMove($neighborhood\_operator$, $s(t)$);
5: AddToNeighbor($s'(t)$,$n(s)$);
6: SelectSolution($s(t+1)$, $n(s)$);
7: $t$:=$t+1$;
8: **until** $Terminaison\_criterion$

**Algorithm 2** The general template of a hill climbing algorithm

1: Initial Solution ($s_0$)
2: $t := 0$;
3: **repeat**
4: $s'(t)$:= ApplyMove($neighborhood\_operator$, $s(t)$);
5: AddNeighbor($s'(t)$, $n(s)$);
6: SelectBestSolution($s(t+1)$, $n(s)$);
7: $t$:=$t+1$;
8: **until** $Local\_optimum\_reached$

Other types of S-meheuristics exist like simulated annealing, iterative local search

and variable neighborhood search. More details about those approaches are presented as a survey and a state-of-the-art on S-metaheuristics in [Talbi 2009].

**P-Metaheuristics** Population-based metaheuristic algorithms (P-metaheurstics) share a lot of concepts together. They also proved to perform very well when addressing complex problems for academic or industry purposes. The principle of P-metaheuristics relies on the evolution of an initial population of solutions through iterations in order to generate a new population that will replace the previous one. The generation of the new population from the current population uses operators. The selection of the individuals of the population that will generate the future individuals follows different policies. The algorithm iterates that way until reaching the stopping criterion (see Algorithm 3). Among the best known P-metaheuristic algorithms are evolutionary algorithms (EAs), scatter search (SS), estimation of distribution algorithms (EDAs), particle swarm optimization (PSO), bee colony (BC), etc. More details about this class of metaheuristics or about the cited algorithms can be found in [Talbi 2009]. In this manuscript we deal mainly with evolutionary algorithms (see Algorithm 4), more specifically genetic algorithms. Those types of algorithms are stochastic, that means that they use operators for evolution that conduct to different results from a run to another. The population contains a number of encoded individuals, where each one represents a potential solution. The first population (initial population) is usually generated randomly. Each iteration of the algorithm is called a generation. During a generation a set of solutions is selected. Those selected solutions are recombined using the evolution operators to provide new solutions. The new solutions replace following a certain policy the worst solutions of the previous population. The algorithm relies on the fitness of the solutions to carry out this operation. The fitness is computed with an evaluation function. This operation is repeated until reaching the termination criterion.

**Algorithm 3** The general template of a P-metaheuristic algorithm

1: Initial Population ($P_0$)
2: $t := 0$;
3: **repeat**
4: $\quad P'(t)$:= Generate($P(t)$);
5: $\quad P(t+1)$:= Replace($P(t)$, $P'(t)$);
6: $\quad t$:=$t+1$;
7: **until** $Termination\_criterion$

#### 1.3.2.3 ParadisEO Framework

The ParadisEO framework [3] [Cahon 2004] is an evolution of the EO[4] (evolving objects) [Keijzer 2002]. ParadisEO is presented as white-box object-oriented software

[3] http://paradiseo.gforge.inria.fr
[4] http://eodev.sourceforge.net

**Algorithm 4** The general template of a evolutionary algorithm

1: Initial population ($P_0$)
2: $t := 0$;
3: **repeat**
4: $P'(t)$:= Selection($P(t)$);
5: $P'(t)$:= ApplyEvolutionOperators($P'(t)$);
6: $P(t+1)$:= Replace($P(t)$, $P'(t)$);
7: $t$:=$t+1$;
8: **until** $Termination\_criterion$

framework dedicated to the flexible design of metaheuristics for optimization problems. This framework is based on the C++ library and designed to be portable across different operating systems.

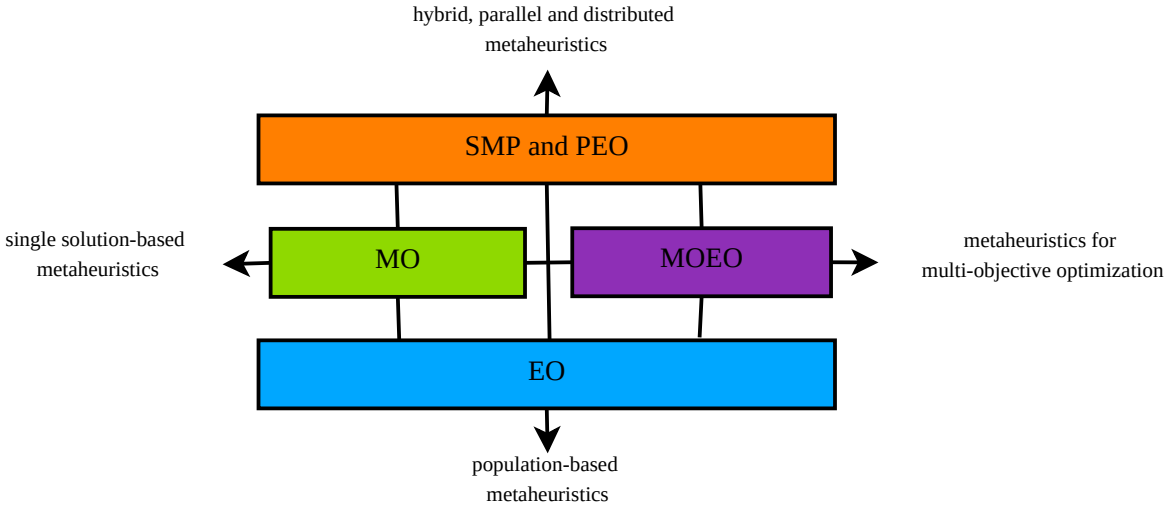



Figure 1.6: Different modules that compose ParadisEO.

As shown in Figure 1.6, the framework is composed of four modules. The idea of this framework is to be as seamless as possible by separating each dedicated metaheuristics module from the problems they are intended to solve. Such a separation provides a maximum code and design reuse to the user.

- **ParadisEO-EO (Evolving Object)** provides the main classes for the development of single-objective P-metaheuristics, including evolutionary algorithms and particle swarm optimization techniques.
- **ParadisEO-MO (Moving Object)** [Humeau 2013] contains the necessary tools for S-metaheuristics such as hill climbing, simulated annealing or tabu search.
- **ParadisEO-MOEO (Multi-Objective EO)** [Liefooghe 2009] is dedicated to the design of multi-objective metaheuristics mainly for the P-

metaheuristics. It provides the most-common Pareto-based multi-objective tools to use famous multi-objective algorithms such as (IBEA, NSGA, NSGAII, SPEA, SPEA2, ...) or to design its own multi-objective algorithm. It proposes for that a wide range of features such as non-dominated solutions storage, elitism and performance metrics. To do so, one has to instantiate or to override the associated ParadisEO components to achieve his/her purposes.

- **ParadisEO-PEO (Parallel EO)** [Cahon 2004] contains a powerful set of classes for the design of parallel and distributed metaheuristics. It is mainly used to speed up the previously cited modules by providing for instance parallel evaluation of solutions, parallel evaluation of the objective function and parallel cooperative algorithms.

All the proposed approaches in this thesis except the work in Chapter 4 have been implemented using mainly the MOEO module of the ParadisEO framework.

## 1.4 (Meta)heuristics for Cloud Scheduling

### 1.4.1 Cloud scheduling: problem formulation and models

#### 1.4.1.1 Problem formulation

The scheduling problem is a famous problem in the literature [Rajpathak 2001]. However, its formulation in the field of cloud computing is not obvious. There are different entities in the cloud scheduling model depending on the scheduling level of the approach (see Section 1.4.1.2). Thus, when dealing with the service-level scheduling, two types of entities are considered: the client service requests and the cloud providers. For the task-level scheduling, one considers a set of client applications and a set of VMs instance types. And for the last type, the VM-level scheduling, the actors of the assignment process are respectively a set of VMs and the set of the hosts that compose the cloud (see Figure 1.7). One can notice that the objective for all the scheduling levels is the same, allocating the first set (client service requests, client applications, VMs) of $J$ items on the second set (cloud providers, VMs instance types, hosts of the cloud) of $N$ items. We can then formally define the scheduling problem on the cloud with its constraints and objective functions as shown in the Equations (1.2), (1.3), (1.4).

$$Optimized\ criterion = (\sum_{i}^{N}\sum_{j}^{J} Metric_{ij} \times x_{ij}) \times Time \tag{1.2}$$

$$Constraint\ criterion = ((\sum_{i}^{N}\sum_{j}^{J} Metric_{ij} \times x_{ij}) \times Time) \leq Constraint\ value \tag{1.3}$$

$$\sum_{j}^{J} x_{ij} = 1 \ for \ each \ i, 1 \leq i \leq N \tag{1.4}$$

Where $Metric_{ij}$ represents the metric value of the optimized criterion where a member $i$ of the set $N$ is assigned to a member $j$ of $J$. The $Time$ value represents the time duration of the assignment and the $Constraint\ value$ value the bounded limit of the optimized criterion (e.g. deadline). $x_{ij}$ represents the placement constraints, where each member $j$ of the set $J$ can be assigned to one and only one member $i$ of the set $N$.

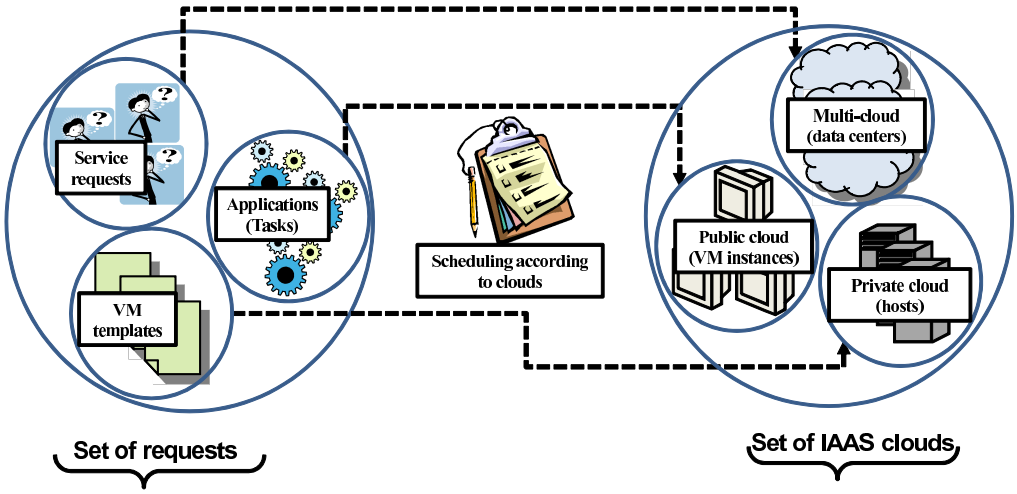



Figure 1.7: Different cloud scheduling levels depending on the couples (scheduled entities, type of IAAS cloud).

#### 1.4.1.2 Scheduling models

Cloud computing as a parallel and/or distributed architecture is necessarily faced to scheduling. This scheduling can affect the performance of the cloud in a right or a wrong way. Therefore, it is critical for a cloud to include an efficient scheduler for optimizing different criteria depending on the providers needs. The scheduling process can be done at different levels depending on the nature of the proposed service. Therefore, according to whether the service of the cloud is market-oriented or not-market-oriented, the addressed criteria and the hierarchical level of scheduling will not be the same. We identified three levels of scheduling. As shown in Figure 1.8, two of them are market-oriented (Service-level scheduling and Task-level scheduling), and one is not-market-oriented (VM-level scheduling). We present the characteristics of each level of scheduling in clouds in the following.

- *Service-level scheduling:* this level of scheduling is static and concerns a part of the resource management layer. The aim of this type of scheduling is to allocate coarse-grained requests which represent services.

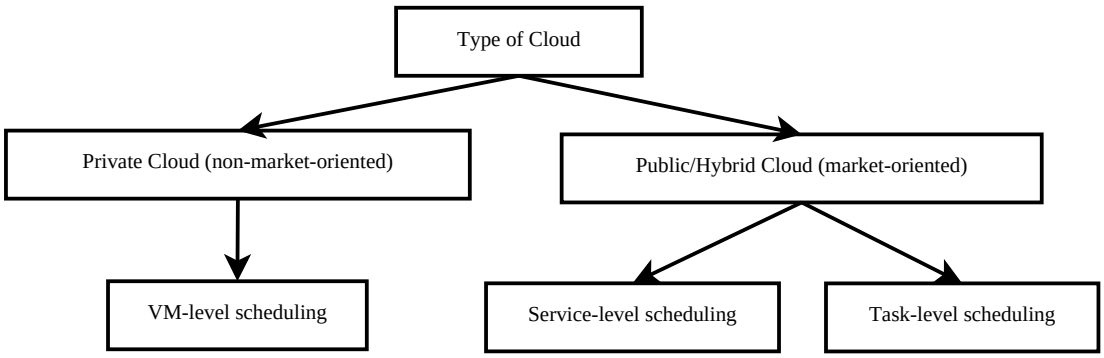



Figure 1.8: Different scheduling levels according to the type of clouds.

In this high level scheduling type, the cloud architecture is composed of several geographically distributed clouds. The scheduler has to be aware of the general architecture configuration. A global view is needed to let the scheduler assigning the workflows over the different clouds. Such scheduler is called *meta-scheduler* and proceeds in three steps. The first step is not mandatory depending on the number of requested services. Based on functional parameters, it is responsible to find the collection of clouds that best fits the requested service in the cloud market. If no cloud proposing that service is found, the user can be asked to upload his/her own environment for running his/her service. The second step is related to the granularity of the QoS requested by the customers.

Indeed, depending on the fine- or coarse-grained QoS, the meta-scheduler could need to deduce non-functional parameter (time, cost, ...) from functional set of parameters. Once the step two done if necessary, the last step is introduced to allocate the requested service using the fined-grained QoS by making the reservation on the appropriate cloud. The meta-scheduler can estimate at that point the obtained values for the users' requirements according to the chosen assignment. The contract between the user and the provider is signed and represented by the service level agreement (SLA).

Moreover as said previously, this assignment is static. Thus, it is necessary to propose local schedulers within each data center (cloud) to be able to adapt to the system evolution such as resource volatility and resource overload. Therefore, a dynamic scheduling has to be provided, this is offered by the *task-level scheduling*.

- *Task-level scheduling:* the task-level scheduling addresses a lower resource management layer than the service-level scheduling. This type of scheduling is dynamic and adapts itself to the cloud changes. Its aim is to optimize the task-to-VM assignment according to the QoS constraints of each task of each customer, while minimizing the overall running price and time costs. Unlike the meta-scheduler, the task-level scheduler is dedicated to a single data

center (cloud), it is unable to manage the resources of other clouds of other providers. The matching process between the task and the VM is done by a third-party called *broker*.

Moreover, as the service-level scheduler, the task-level scheduler has three deferent steps in its process. At the first step, the scheduler aims to calibrate the task QoS constraints. Indeed, for instance, when a workload is scheduled at the service-level, all the tasks are sent together to the cloud which is selected by the meta-scheduler. However, these tasks will not be handled at the same time by the task-level scheduler. Therefore, new constraint values have to be given to the task according to the real time when they are going to be processed effectively. The second step consists in the optimization of the matching process between the task and the virtual machines (VMs). This is done dynamically according to the resources needs, to the SLA, to the price and the availability of the VMs.

The VMs are created on demand by booking them to the provider to suit the previously cited needs. The problem can even be more dynamic and complicated when the price of the same VM instances can change during the time, we call that VM spots. That leads the scheduler to change the assignment of each task continuously depending on the VM price and the general load to optimize the QoS. In general, the tackled objectives for the QoS in this type of scheduling are both the overall response time and cost related to the task processing. The aim is to reduce them both while providing an interesting profit to the broker. The last step, consists in the implementation of the optimal or near-optimal scheduling over the architecture after being computed by the step two. Note that both service-level scheduling and task-level scheduling can be complementary and belong to the market-oriented cloud.

- *VM-level scheduling:* the VM-level scheduling is the lowest scheduling level, used to provide the VMs that are requested by the task in the task-level scheduling. This scheduling manages the VMs of a cloud running under a cloud manager (see Section 1.2.2) by using the virtualization possibilities offered by the hypervisor.

  The aim is to find the best VM scheduling over the hosts that compose the data center (cloud). This is done according to the VM features in terms of resources to respect the constraints of feasibility, while optimizing the cloud provider objectives. This type of scheduling is inherited from the grid infrastructure replacing the VMs by jobs. Thus, unlike the service-level scheduling and the task-level scheduling, the VM-level scheduling is not market-oriented approach. In other words, there is no reference to profit at this level of scheduling.

  The objectives are exclusively oriented towards the performance improvement of both the data center energy consumption and the VMs response time. The

scheduling process is performed in three steps. At the first step of the VM-level scheduling, the scheduler retrieves the VM characteristics according to the specified needs (e.g. a task that needs a VM with a certain amount of CPU and memory) and using the hypervisor to retrieve the hosts features of the cloud.

Once this done, it filters both the list of VMs and hosts keeping only the available hosts and the deployable VMs. The second step consists in assigning the pool of deployable VMs to the available hosts. This assignment is done with respect to the VM constraints and the hosts' capacities. In addition, as said previously the assignment has to be optimal or near-optimal regarding the objectives. At the last step, the best scheduling is validated and the VM deployed on the infrastructure. The hosts' capacities are updated by the hypervisor waiting for other VMs. Note that the problem evolves in the time.

Indeed, each time a VM finishes its processing new assignment possibilities can emerge. Moreover, it is possible to make the problem fully dynamic by moving the VM from hosts to hosts while still processing. This is called live migration. However, this technique faces some issues related to the flexibility of the applications running on the VM (interruptibility, user rights, workspace, ...), the data transfer cost and the CPU time complexity of the VMs. It also causes none negligible scheduling overhead that has to be taken into account.

### 1.4.2 Related Works

#### 1.4.2.1 Scheduling on market-oriented clouds

**Service-level scheduling** The ancestor of the market-oriented cloud computing can be found in a concept called *utility computing*. The works such as [Chun 2002, Irwin 2004] are among the firsts that gave birth to utility computing model in the field of computer science. Thus, the common criterion of all the approaches that results from this model treats about economics. The work in [Lee 2010a], where two algorithms based on a pricing model are proposed, clearly illustrates this point. Both algorithms use processor sharing in order to balance between conflicting objectives: profit and response time. In [Burge 2007], 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 [Yu 2006] and [Garg 2008]. In [Garg 2008], a linear programming driven genetic algorithm is proposed. 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 [Yu 2006] present a genetic algorithm approach to address scheduling optimization problems in workflow applications with two QoS constraints (deadline and budget). All the previously cited works refer to the strict utility computing model or to a cloud computing model resulting straightly from it. In this type of model, the interaction is done directly between the client and the provider. That

forms a bi-polar model with two tiers i.e. the client and the provider.

However, the marketing of cloud computing paradigm is being more and more complex and raises a number of issues. Indeed, when one mentions the performance or the QoS, one necessary implies expensiveness, mainly when dealing with cloud models where the provider has a full pricing control. To deal with such an issue, an evolution of the model has been introduced. In the latter, a third tier appears and plays the role of an intermediate to find the tradeoff between customers' needs and the providers' profits. A set of works have been conducted over this new model. The work in [Chaisiri 2009], proposes an optimal virtual machine placement algorithm. The objective is to minimize the costs while assigning VMs in a multiple cloud provider environment. For that, they use a strategy that avoids the two extreme (over and under) assignments. The approach fits the correct resources needs by adjusting the pricing according to the load arrival of VMs.

Other works like [Tordsson 2012] and [Lucas-Simarro 2012] give a good illustration of the cloud broking applied to the service-level scheduling. In [Tordsson 2012], 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 using a broker affects beneficially the performances and lowers the costs of the deployed services. The same goes for [Lucas-Simarro 2012] where the work focuses on different scheduling strategies for an optimal deployment of virtual services across multiple clouds addressing (e.g. budget, performance, instance types, placement, etc.) in an aggregate way. They study different optimization criteria, constraints and environmental conditions.

In [Elmroth 2009], the authors outline usage scenarios and a set of requirements for a federation of cloud infrastructures based on RESERVOIR. They also propose an accounting and billing architecture between resource consumers and infrastructure providers to be used within RESERVOIR. The objective is to cope with the migration of virtual machines by managing postpaid and prepaid payment schemes according to the users' needs.

All of the last presented approaches take into account the profit and/or other objectives in their study but they do not consider the relationship between the dealt with criteria. They also do not pay attention on the energy consumption of their architecture. The work presented by Garg *et al.* in [Garg 2010] deals with those points, by proposing a new energy model for service-level scheduling that includes gas emissions and pricing. Several heuristics are proposed to find a good tradeoff between the objectives. However, this approach is a prioritization of objectives i.e. it can only optimize one objective at a time.

We notice through all the presented work over the service-level scheduling two major issues. The first is the lack of energy consideration in many works, despite the crucial importance that may have the energy over such huge distributed architecture. The second point is about the multi-objective approach used in those works. It is always a pseudo multi-objective approach, i.e. the objectives are aggregated or ranked in a decreasing order from the most to the less important. However, this order is subjective and changes according to the needs. In addition, switching the

objective order can totally change the results and complicate the results analysis. In the work that we propose, both a Pareto multi-objective approach and an energy model are considered to deal with the service-level scheduling.

**Task-level scheduling** As said earlier, the second part of the market-oriented cloud computing in addition to the service-level scheduling is the task-level scheduling. Unlike the service-level scheduling, the aim of the task-level scheduling is to address the task assignment within each cloud. The work presented by *Chen et al.* in [Chen 2011] proposes an utility theory based on economics to develop a new model that integrates the client satisfaction over a cloud. Two scheduling algorithms are proposed to find a good tradeoff between the client satisfaction and the provider profit during the task assignment. However, this approach is an aggregation of objectives (i.e. it combines several objectives to create a new one). The work in [Wu 2013] proposes a market-oriented package for a hierarchical scheduling in a distributed cloud. The approach deals hierarchically with both the service-level scheduling and the task-level scheduling where individual workflow instances are mapped to cloud services in a multi-provider cloud. The objectives of this approach consist in minimizing the makespan, cost and CPU time. It uses for that a random algorithm for the service-level scheduling and three metaheuristics (Genetic Algorithm, Ant Colony Optimization and a Particle Swarm Optimization) for the task-level scheduling.

All the above surveyed works, present different brokering approaches to deal with a task-level scheduling. However, none of those works consider the relationship between the charged price and the provided performance. As for the service-level scheduling, they also do not pay attention on how each of those criteria can affect the others, and no one of them tackles them simultaneously. This issue is also addressed through a new Pareto economic model for task-level scheduling in this thesis.

#### 1.4.2.2 Scheduling on non-market-oriented clouds

After a race to performance and profit, utility and cloud computing paradigms are facing an energy problem. To reduce energy consumption, various issues such as resource management in both software and hardware layers must be addressed. Software approaches are mainly based on virtualization or task consolidation. The consolidation technique aims to maximize resource utilization by minimizing energy consumption. Indeed, a resource allocation strategy that takes into account resource utilization should lead to better energy efficiency. Hardware approaches use the opportunity offered by manufacturers in modern processors to adjust the voltage and frequency. This adjustment can vary the processor performance and thus its energy consumption.

Therefore, several works have been proposed in the field of the energy-aware computing. Those approaches are based either on software or hardware layer of the system (i.e. system-level) (see Figure 1.9).

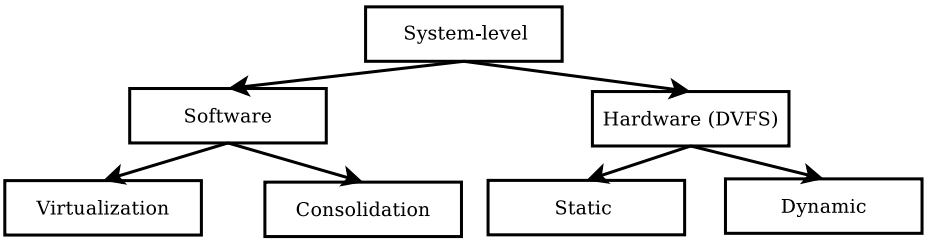



Figure 1.9: Classification of system-level energy reduction techniques.

Before the generalization of the virtualization, most of the energy-aware approaches related to the system-level were not generic, tackling this issue only by referring and focusing on scheduling dedicated applications (tasks) over specific architecture. Those methods belong to the hardware branch of the system-level tree. In [Lee 2009, Rizvandi 2010] for example a hardware technique (DVFS) is proposed. It consists in dynamically varying the CPU frequency in order to minimize the energy consumption. Moreover, in [Guzek 2012], the authors investigate the influence of the DVFS using the multi-objective evolutionary algorithms (NSGAII) for bi-objective scheduling (makespan and energy consumption) of a Directed Acyclic task Graph (DAG) on heterogeneous multi-processor platform. The drawback of this type of methods is the assumption about the existence of a tight coupling between the tasks and the resources.

To tackle this issue, other methods using consolidation appears in the system-level. In [Lee 2010b], 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 both computation and idle energy consumption. 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 a performance degradation of that task. However, consolidation suffers also from issues. The paper [Srikantaiah 2008] exposes some of these issues mainly regarding the power optimization purposes and proposes some research directions to address the involved challenges. Moreover, another way to reduce cloud computing energy footprints is proposed in [Tesauro 2007]. The author presents a reinforcement learning approach to deal with the optimization of two main criteria, performance and power consumption.

However, because of the evolution of the architectures due to the generalization of the virtualization concept, the service-level scheduling is becoming exclusively a VM-level one. Since inheriting from the system-level, the VM-level scheduling follows the same tree as drawn in Figure 1.9. In this case, the hardware branch becomes less close to the hardware. Thus, it applies techniques such as DVFS for scheduling VMs instead of dedicated applications. In addition, in the VM-level scheduling, some of the previous methods like consolidation evolved from scheduling tasks to scheduling VMs.

In [Mezmaz 2011], we proposed a preliminary work for the transition step from

the system-level scheduling to the scheduling on cloud infrastructure (VM-level). In this work, we investigate the problem of scheduling precedence-constrained parallel applications on heterogeneous computing systems (HCSs) like cloud computing infrastructures. This kind of application was studied and used in many research works. We propose a new parallel bi-objective hybrid genetic algorithm that takes into account, not only makespan, but also energy consumption. We particularly focus on the island parallel model and the multistart parallel model to improve the existing results. Our new method showed the potential of the dynamic voltage scaling (DVS) to minimize energy consumption. More details about this work are presented in Appendix B.

**VM-level scheduling** One of the most striking evolution between the VM-level and the system-level can be found in the work proposed in [Laszewski 2009]. Thus, in this work, 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 [Laszewski 2009] is that it deals only with one cluster and not with a large number of machines that compose in general a cloud. In addition, this work makes an assumption on the hardware configuration of the machines that composes the cloud, assuming that they are equipped with DVFS.

In the work presented in [Andreolini 2010], the proposed algorithm aims to reallocate virtual machines in a large scale cloud. The approach consists in identifying only the critical instances using the load trend behavior instead of the thresholds to take decisions in order to load balance the clouds. The contribution of this work is an improvement in the selectivity and the robustness of the migration process. However, no interest is given to the energy criterion.

Some other works model the scheduling problem of VMs as a bin packing where the VMs are the objects to pack and the machines the boxes. [Verma 2008] 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. Moreover, the work in [Borgetto 2012b] presents an approach using VM migration and reconfiguration, and physical machine power management for energy reduction purposes while offering a correct SLA. It is based on an autonomic management loop using different heuristics including FFD based one.

The FFD and its variants have been proved in a comparing study [Mills 2011] 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 find the global optimum (best solution) only for landscapes with a single local optimum. Nevertheless, the scheduling problem changes through the different VMs arrival, and the solution landscape changes as well creating different local optima. It is then necessary to explore other local optima using metaheuristics such as those we

propose in this thesis.

Another work dealing with both power consumption and job performance is proposed in [Borgetto 2012a]. The objective of this paper is to find the best allocation of job services over infrastructures using three different formalizations to the problem using for each formalization heuristics to find the best tradeoff between the objectives. The main issue with such a formalization is that the problem is not Pareto multi-objective anymore. It becomes either a lexicographic or an aggregation problem.

Another method consists to propose a native energy-aware cloud manager. Snooze [Feller 2012] for example 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. This technique is the evolution of the work proposed in [Orgerie 2008]. The author deals in the latter with energy consumption reduction in large-scale computational grids like Grid'5000 by switching off idle nodes in a clever way. One can see the impact of the virtualization on the concept replacing the nodes by the VMs. The major drawback of such techniques is the energy consumption extra cost caused in the case of poor anticipation while switching off the machines/VMs.

Moreover, it is rare to find multi-objective approaches addressing the VM scheduling like the work [Xu 2010]. Indeed, this work proposes a VM placement policy using a genetic algorithm relying on a fuzzy multi-objective evaluation to simultaneously minimize the total resource wastage, the power consumption and the thermal dissipation cost. The major drawback is that, unlike a real Pareto set approach that offers diversity in the resulting solutions by proposing a wide choice of non-dominated solutions, using the proposed fuzzy multi-objective technique provides a unique final solution.

To deal with all the misses mentioned before such as, the lack of diversity of the heuristics approaches, the hardware assumptions like DVFS in the models and the negligence of the impact of the energy reduction over the VMs performance, we propose a work in this thesis, where all the aforementioned issues are addressed with a Pareto VM-level scheduler embedded into a cloud manager (OpenNebula). Embedding the scheduler offers the portability related to OpenNebula and the implicit management of the cloud constraints (e.g. scheduling time limit, assignment correctness, etc.) since it uses the Grid5000 platform as an IAAS infrastructure.

Table 1.2: Classification of the related work.

| Approach | Service-level scheduling | Task-level scheduling | VM-level scheduling | Market-oriented cloud | Cloud manager embbeded | Energy-aware model | Performance-aware model | Metaheuristics algorithm | Pareto optimization |
|---|---|---|---|---|---|---|---|---|---|
| Xu and Fortes [Xu 2010] | no | no | yes | no | no | yes | no | yes | yes |
| Feller et al. [Feller 2012] | no | no | yes | no | yes | yes | no | no | no |
| Verma et al. [Verma 2008] | no | no | yes | no | no | yes | yes | no | no |
| Andreolini et al. [Andreolini 2010] | no | no | yes | no | no | no | yes | no | no |
| Von Laszewski et al. [Laszewski 2009] | no | no | yes | no | yes | yes | no | no | no |
| Rizvandi et al. [Rizvandi 2010] | no | no | yes | no | no | yes | yes | no | no |
| Lee and Zomaya [Lee 2010b] | no | no | yes | no | no | yes | no | no | no |
| Wu et al. [Wu 2013] | yes | yes | no | yes | yes | no | yes | yes | no |
| Chen et al. [Chen 2011] | no | yes | no | yes | no | no | yes | no | no |
| Elmroth et al. [Elmroth 2009] | yes | no | no | yes | yes | no | no | no | no |
| Lucas-Simarro et al. [Lucas-Simarro 2012] | yes | no | no | yes | no | no | yes | no | no |
| Tordsson et al. [Tordsson 2012] | yes | no | no | yes | no | no | yes | no | no |
| Lee et al. [Lee 2010a] | yes | no | no | yes | no | no | yes | no | no |
| Burge et al. [Burge 2007] | yes | no | no | yes | no | no | no | no | no |
| Yu and Buyya [Yu 2006] | yes | no | no | yes | no | no | yes | yes | no |
| Garg et al. [Garg 2008] | yes | no | no | yes | no | no | no | yes | no |
| Garg et al. [Garg 2010] | yes | no | no | yes | no | yes | no | no | no |
| Our work | yes | yes | yes | yes | yes | yes | yes | yes | yes |

## 1.5 Conclusion

In this chapter, we have presented all the concepts and prerequisites needed to the general comprehension of our contributions. Thus, we have emphasized the different level of scheduling related to the cloud computing architecture and proposed a general formulation for cloud scheduling purposes. We also have described the metaheuristics techniques within the optimization context with a special focus on the multi-objective Pareto metaheuristics. Last but not the least, we have highlighted the challenges of this thesis by depicting the major lacks of the different works of the literature, regarding both the used optimization technique and/or the cloud orientation policy. We summarize them in the following:

- *Pareto-based:* despite the multi-objective nature of scheduling over a cloud computing architecture, the major works presented in the literature are even mono-objective or based on an aggregation of objectives. Moreover, we have shown the significance of the energy in this type of architectures and the relative lack of consideration regarding this criterion. Thus, we have addressed all those misses in this document by proposing each time Pareto approaches dealing with conflicting objectives.

- *Multi-level scheduling* due to the complexity of the cloud computing concept, we have shown that the scheduling process is different according to three major levels. This classification depends on the granularity of the user's service requests and/or the market-orientation policy of the cloud. Therefore, we have proposed in this manuscript an approach for each aforementioned case to address all the possible needs.

- *Feasibility:* all the different works on the cloud computing topic are mainly based on artificial models and experimentations. However, the ultimate goal is to propose a ready to use approach. In addition, the challenge of dealing with an optimized scheduling in cloud computing involves necessary the validation of the approach by facing it to the real world constraints. This can be done only by deploying the algorithm over a real and physical infrastructure. This is what we have done in this manuscript by integrating a Pareto multi-objective scheduler within the OpenNebula cloud manager and validating its design by deploying it over a cloud composed of several physical machines.

Chapter 2

# Service-level Scheduling on a Cloud Federation

### Main publications related to this chapter

- Y. Kessaci, N. Melab and E-G. Talbi, A Pareto-based Metaheuristic for Scheduling HPC Applications on a Geographically Distributed Cloud Federation, *Cluster Computing Journal*, 2012.

- Y. Kessaci, N. Melab, E-G. Talbi. A Pareto-based GA for Scheduling HPC Applications on Distributed Cloud Infrastructures. *International Conference on High Performance Computing and Simulation (HPCS)*, Istanbul, Turkey, 2011.

### Contents

## 2.1 Introduction

As said before, cloud computing appears nowadays to be increasingly adopted in many areas by proposing several services. The field of high performance computing (HPC) where HPC infrastructures are proposed as a service, considered here as a case study, does not derogate to this rule. In Section 1.4.2.1 we have observed that for the service-level scheduling the main optimized criterion is the profit. However, the type of the proposed computers in this case consumes a significant and growing portion of energy. Therefore, energy-aware scheduling is crucial for large-scale systems that consume considerable amount of energy.

A recent study [Koomey 2008] 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)[Eff 2011].

On the other hand, green house gas emission is reaching a critical limit. A recent work [Gartner 2007] 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 impact affecting the profit of the providers. Indeed, according to Amazon's estimate [Hamilton 2009], the energy-related costs amount represents 42% of the total data center budget, and includes both direct power consumption for 19% and cooling infrastructure for 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.

Many drawbacks have been mentioned in Section 1.4.2.1. The proposed work in this chapter differs from the previous studies in plenty aspects. First, it tackles the energy issue for this level of scheduling (service-level) and deals with both computing and cooling energy consumptions in the proposed energy model. Moreover, it considers the relationship between energy, green house gas emissions and profit and pays attention on how each one of those criteria can affect the others. In this purpose, 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 to find the trade-off between all the considered criteria. Finally, the experiments in our work are realistic and performed on a long period of workloads composed of heterogeneous HPC applications in order to avoid the tightly coupled applications issue.

The remainder of this chapter is organized as follows. In Section 2.2 we present the application, system and energy models used in our problem modeling. Our approach is presented in Section 2.3. The results of our experimental study are

reported and discussed in Section 2.4. The conclusion is drawn in Section 2.5.

## 2.2 Service-level Scheduling Model

### 2.2.1 Cloud model

The type of cloud considered in this chapter is a public IAAS federation cloud. More precisely, we deal with a two-tier market-oriented HPC service cloud. In the first tier we find the provider and on the other tier, the clients. The model of our architecture is a cloud federation which is geographically distributed over the world and inspired from the Open Cirrus project [Campbell 2009]. The clients have access to the cloud by requesting resources to the provider (see Figure 2.1). The service proposed by the cloud provider in our approach offers infrastructures to the clients in order to run HPC applications. As dealing with service-level scheduling over a cloud federation, the scheduler needs to have an overall view (meta-view) of the cloud federation. Thus, the originality of this approach consists in a meta-scheduling algorithm that uses a multi-objective genetic algorithm in order to find the best meta-scheduling to the requests 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 of the federation 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/her profit, the meta-scheduler algorithm aims also to give the best Quality of Service (QoS) to the client by meeting the maximum applications deadlines and respecting the model's constraints.

The optimization of the objectives can be achieved through the exploitation of 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 distributed clouds and the gas emissions, both due to the used policies 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 tier of our model is detailed in the following:

- **Client side:** as a case study the requests submitted by the distributed cloud's clients are HPC applications. This means that the service is computation-intensive. Hence, we do not pay attention in our work to data transfers. Therefore, our model can deal with plenty of types of services other than HPC as long as they are not communication intensive.

  In our approach, the clients submit HPC requests by informing the meta-scheduler about the time duration of the application execution and the number of needed processors. The information about the time duration of the application can be deductable following two methods. The first method considers the duration time of the execution as the reservation time. Hence, for

his/her interest the client has to reserve enough time for his/her application, otherwise it is aborted. Therefore, the user has sometimes to overestimate the execution time of his/her request and pays for longer than the real execution time of his/her application. The second method is prediction [Iverson 1999]. 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 time slot to satisfy the request by respecting the deadline, the reservation for the application will fail. Each HPC application has to be hosted in one and only one cloud. 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 Chapter 3.

- **Provider side:** in our approach, the provider is the owner of all the clouds of the cloud federation. For instance Amazon [Ama 2012b], one of the world leader cloud providers, 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 each local cloud to get its state 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 characteristics, it will be chosen by the meta-scheduler and its state will be updated. These other characteristics are specific to each cloud, they are given by the local scheduler of each cloud to the meta-scheduler helping it to find the best meta-scheduling. Each cloud's local scheduler has access to the local information such as the execution price (cost), 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.

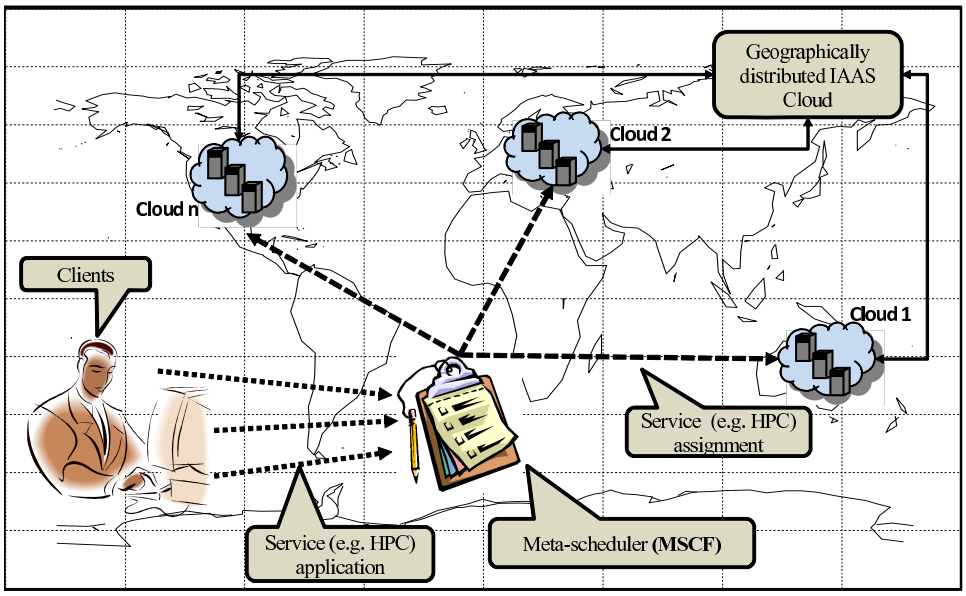



Figure 2.1: A representation of our two-tier cloud federation model.

### 2.2.2 Energy and profit models

As shown in the work presented in [Kliazovich 2012], the energy consumption of a cloud results from IT equipments (network, storage and computing) and auxiliary equipments (lighting, cooling, ...). The infrastructure that we deal with is dedicated to HPC computing. Thus, we do not consider lighting consumption among the auxiliary equipment since its impact is negligible. However, for the same HPC reasons we deal only with computing energy consumption. Indeed, this represents the largest amount of consumed energy by the infrastructure. In our energy model, we also do not pay attention to how the energy is optimized within each cloud, our focus is made over the whole cloud federation.

Our energy model is derived from the power consumption model in Complementary Metal-Oxide Semiconductor (CMOS) logic circuits [Burd 1995, Pillai 2001]. 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:

$$P = ACV^2 f + I_{leak} V + P_{short} \tag{2.1}$$

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 perform voltage switching. Notice that $A$ and $C$ are constant values, let $\alpha$ be their product. The second part of the equation represents the

static consumption, this value is also constant, let it be $\beta$. In CMOS processors the voltage can be expressed as a linear function of frequency [Chen 2005b, Wang 2008], $V^2 f$ is replaced by $f^3$. The new equation becomes:

$$P = \alpha f^3 + \beta \tag{2.2}$$

In addition, another source of energy consumption needs to be taken into account. In fact, the energy used for cooling each cloud of the federation is consequent and has to be integrated in our energy model. Energy dedicated to cooling is tightly related to the geographical area where the cloud is situated since the temperature changes from an area to another. To compute cooling energy amount, each cloud has a coefficient called $COP$ which represents the ratio between the energy dedicated to 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 over time. By using $COP$ the meta-scheduler is able to deduce the energy consumed by each cloud for cooling their devices. This is given by Equation (2.3).

$$E^h = E^c / COP \tag{2.3}$$

where $E^h$ represents the total energy consumed for cooling the cloud 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 geographical location 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 (2.4)).

$$Profit = pr^u - pr^e \tag{2.4}$$

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.

### 2.2.3 Meta-scheduling problem modeling

As previously said in this chapter, we deal with a high level scheduling called meta-scheduling. The objective is to schedule services, in our case HPC services, over a geographically distributed cloud infrastructure. In this sense, the interactions in our model concerns two-tier. 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 saw in the Section 1.1 that the scheduling problem is in general NP-hard [Garey 1979]. Therefore, our large size multi-objective meta-scheduling problem is NP-hard as well. Thus, a metaheuristic algorithm appears to be the most appropriate approach

to tackle this problem.
To be as realistic as possible, the provider has to pay an expenditure for the used cloud $i$ in our model. This expenditure is the result of the electricity consumption during the computation of the hosted HPC service and is noted $p_i^e$ (\$/kW h). As for all the market-oriented clouds, the client has also to pay for the requested service. The provider fixes the clients' price according to $p_i^e$ price. 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 origin of the electricity used to power each cloud (i.e. type of energy used for the electric power generation: fuel, water, nuclear, wind, ...).

During the scheduling process, the client submits a request for an HPC service $j$. A request is defined by a triplet $(e_j, n_j, d_j)$, all the triplet's information are given by the client 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 respectively the execution time (reservation time) of the HPC application $(e_j)$, the number of processors needed by the client for his/her application $(n_j)$ and finally the deadline after what the application will be considered as failed $(d_j)$. Our triplet is inspired from Amazon EC2 [Ama 2012b] except for the duration time of the client's application. As said previously in Section 2.2.1, this parameter depends on the client itself, he/she has sometimes to pay for a longer reservation period to ensure the completion of his/her application, even if the application finishes earlier. In the following, we present the mathematical formulation of our problem and the used functions for computing the fitness of each candidate solution (scheduling).

- Energy consumption of the CPUs is given by:

$$E_{ij}^c = (\alpha_i(f_{ij}^3) + \beta_i) \times n_j e_j \tag{2.5}$$

- From Equation (2.5) and the Coefficient of Performance (COP), the total consumed energy is deduced as:

$$E_{ij} = \frac{COP_i + 1}{COP_i} \times E_{ij}^c \tag{2.6}$$

- The total carbon emission is given by:

$$(CO_2E)_{ij} = r_i^{CO_2} \times E_{ij} \tag{2.7}$$

- The profit is given by:

$$(Profit)_{ij} = n_j e_j p^c - C_{ij}^e \tag{2.8}$$

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 expenses that the provider has to pay for the used resources in the cloud $i$ for executing the application $j$.

Equation (2.6) 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 (2.3)

$$\begin{aligned} \mathrm{E} = \quad & E^c + E^h = E^c + \tfrac{E^c}{COP} \\ = \quad & E^c \times (1 + \tfrac{1}{COP}) = \tfrac{COP+1}{COP} \times E^c \end{aligned} \tag{2.9}$$

The objective functions of our approach are formulated as follows:

$$Minimizing\ the\ energy\ consumption = \sum_{i}^{N}\sum_{j}^{J}(E)_{ij} \tag{2.10}$$

$$Minimizing\ Carbon\ Emission = \sum_{i}^{N}\sum_{j}^{J}(CO_2E)_{ij} \tag{2.11}$$

$$Maximizing\ Profit = \sum_{i}^{N}\sum_{j}^{J}(Profit)_{ij} \tag{2.12}$$

with the following constraints:

- The application $j$ has to finish before $d_j$, otherwise the scheduling is invalidated,
- 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 Equation (2.10), to reduce the distributed cloud's carbon emissions for Equation (2.11) and to maximize provider's profit for Equation (2.12). Equation (2.11) could be wrongly considered similar to Equation (2.10), 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 awareness is not necessary good for the $CO_2$ emissions. To prove the truthfulness of the aforementioned assumption, we have performed a correlation test between those two objectives. The correlation coefficient between those two objectives on a sample of 1000 solutions is 0.57.

## 2.3 The Meta-scheduling Proposed Approach (MSCF)

In this section, we describe in details the steps of our Meta-Scheduler for Cloud Federation (MSCF) including the multi-objective genetic algorithm (MO-GA) proposed in this chapter.

### 2.3.1 Steps of the MSCF algorithm

Before each scheduling, the meta-scheduler (MSCF) waits a fixed period of time called *scheduling cycle*. This period helps to gather a pool of applications in order to get a larger choice and thus, optimize the scheduling. Once this phase done, the pool is managed by MO-GA to find the best possible assignments over the different clouds which compose the federation. The result of the execution is stored as a Pareto archive. Once the set of Pareto solutions (assignments) is proposed, the algorithm chooses one scheduling according to the provider's decision. The chosen solution from the Pareto set is used as the new state of the cloud federation. This state will be a basis from which the next iteration of the algorithm will make another processing on a new pool of applications. The algorithm keeps iterating and proposes assignments for each new pool of applications (see Figure 2.2).

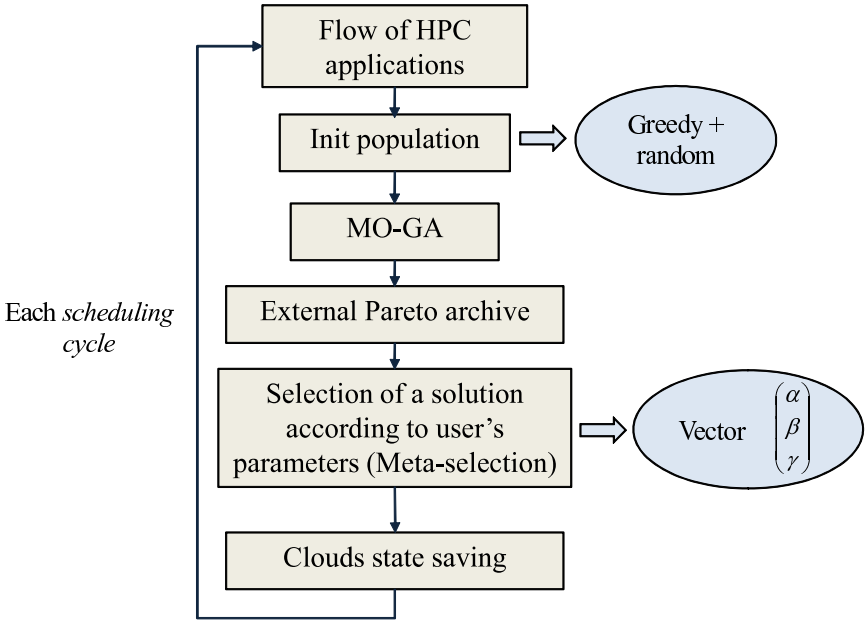



Figure 2.2: A flowchart representing the meta-scheduling (MSCF) algorithm's steps.

### 2.3.2 Service-level scheduling 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 Figure 2.3).

Figure 2.3 represents one possible scheduling among plenty that proposes the genetic algorithm. This scheduling is the result of processing a pool of HPC requests arrived during the last waiting time period presented later and called *scheduling cycle*. In the proposed example, we identify two major information: the indexes of the table depict the HPC applications that are scheduled and the number in each

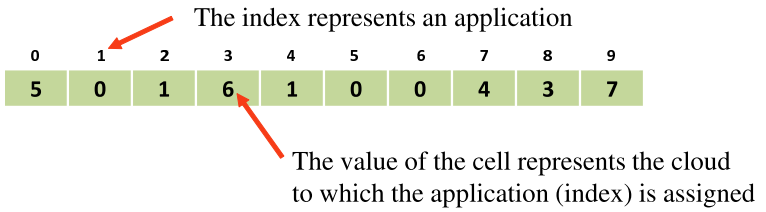



Figure 2.3: A representation of a solution in the meta-scheduling problem (individual in MO-GA's population).

table cell identifies the cloud to which the application is assigned. 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, and helps one to deal with the constraints 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 cannot be handled by the proposed encoding is the deadline constraint. We deal with this constraint in the algorithm by rejecting (invalidating) the assignment that does not respect the deadline of the application. Therefore, at each processing cycle, all the assignments that compose each individual of the MO-GA do not violate any of the model constraint in the current state of the federation cloud (i.e. it exists at least one cloud in the federation, at the current time, which can handle the application in terms of number of processors and respect of the deadline).

### 2.3.3 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 in our algorithm is done according to a combination of two methods. The first method relies on a greedy algorithm and the second follows a random initialization policy. The initialization is decomposed into two steps as follows:

- The first step initializes either one or two element(s) of the population by the result of the greedy method.
- The second step initializes the rest of the population with a random method.

The greedy method reads the applications that arrive during the scheduling cycle and allocates 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 cannot be assigned 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 the whole failure of the scheduling

due to the missing of the deadline of only one application. In other words, this step makes sure that there is at least one feasible solution (scheduling) in the population. It prevents the genetic algorithm from rejecting a big number of applications among the entire pool only because of one unmet application's deadline. As a second step, the rest of the population is initialized randomly. The ratio between the individuals initialized by the greedy algorithm and those initialized by the random method is fixed to 1/15 (i.e. 14 random for each 1 greedy). Coupling with such a ratio both the greedy method and the random method helps to add diversity to the initial population avoiding a bias in the search space of MO-GA. The size of the pool of applications is equal to the total number of applications arrived during the *scheduling cycle* minus the ones eliminated in the initialization phase.

### 2.3.4 MO-GA variation operators

The variation operators allow one to explore the search space. Their principle is stochastic and MO-GA uses two of them: in one hand, the mutation operator which is conventional. Indeed, the operator chooses randomly two integers $i$ and $j$ such that $1 \leq i < j \leq n$. Then, the operator swaps the two applications $i$ and $j$ as illustrated in Figure 2.5. On the other hand, the crossover operator uses two solutions $s1$ and $s2$ to generate two new solutions $s1'$ and $s2'$. The operator picks also two integers from each solution to make the crossover. The full mechanism is explained bellow and illustrated in Figure 2.4. 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, in some cases, only one request is received during the scheduling cycle, thus, no operator can be applied (i.e. only one application to schedule). The diversity is obtained in this case from the number of the individuals of the population resulting from the initialization.

To generate $s1'$, the crossover 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.

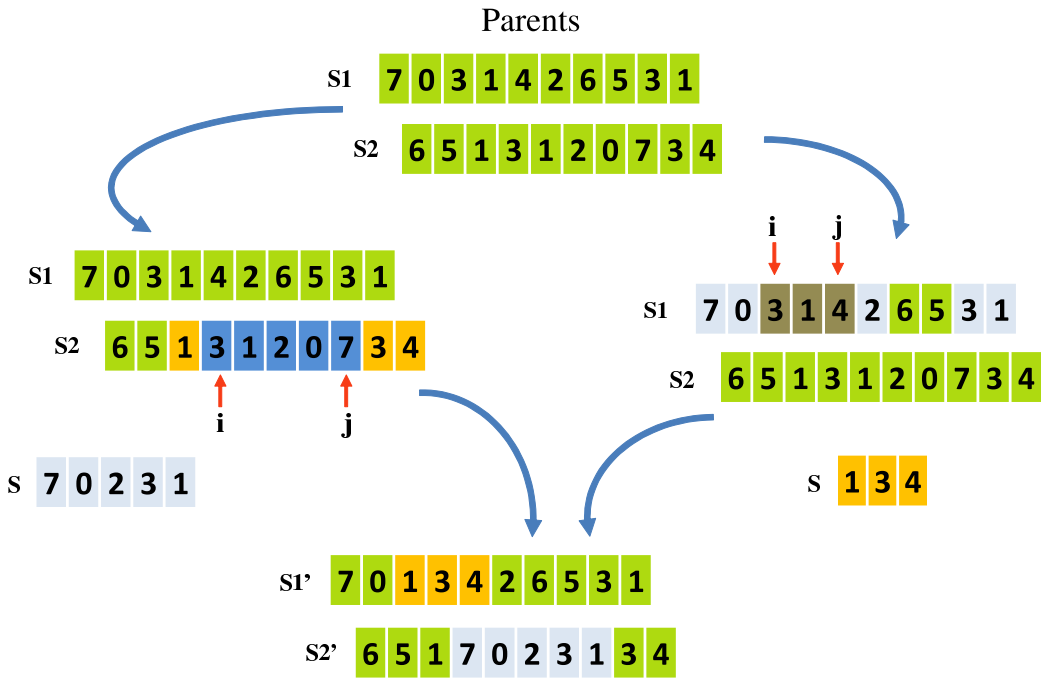



Figure 2.4: The crossover operator mechanism used in MO-GA between two parent solutions $s1$ and $s2$ to generate two offspring solutions $s1'$ and $s2'$.

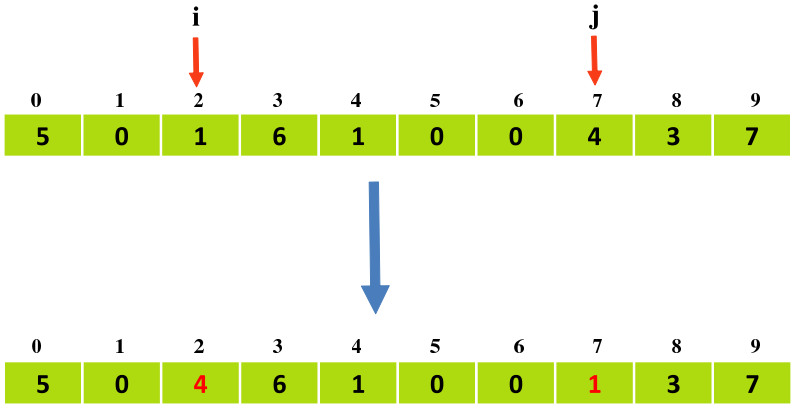



Figure 2.5: The mutation operator mechanism used in MO-GA to reassign two applications by swapping two clouds.

### 2.3.5 Pareto genetic algorithm MO-GA

MO-GA starts by initializing the population as indicated in Section 2.3.3. This population 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 function is called to evaluate the fitness of the offsprings. This fitness is deduced in MO-GA from the energy consumption, $CO_2$ emissions and the generated profit of each scheduling (solution) (see Section 2.2.3). Because of the multi-objective context, the used method in the MO-GA to rank the individuals of the population 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 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. In other words, the Pareto archive is updated at each generation and used by the selection process. The individuals on whom the variation operators are applied, are in the first step, selected according to their elitist 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 to maintain 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 NSGAII algorithm [Deb 2002]. More details about these techniques are given in [Talbi 2009].

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 iterations reached, the external Pareto archive of the MSCF is updated by the last Pareto archive of the MO-GA. The pseudo code of the MO-GA algorithm is presented in Algorithm 5.

### 2.3.6 Provider's policy selection

Marketing is known to be dynamic, changing according to the trends and the permanent evolution of the client's needs. Therefore, one must have several propositions of solutions to fit all these needs. This is the main reason that leads us to propose

**Algorithm 5** Pareto MO-GA algorithm.

1: **Input:** An updated set of cloud and a set of arrived HPC application
2: InitPopulation(Clouds, HPC application, pop);
3: Fitness(pop);
4: DominanceRanking(pop);
5: Calculate the crowding distance;
6: Selection(pop, parent1, parent2) ;
7: Crossover(parent1, parent2, offspring1, offspring2) and/or Mutation(parent1);
8: **if** Check deadline meeting **then**
9: Replace(offspring1, offspring2, pop);
10: **Output:** A set of Pareto non-dominated solutions (assignments)

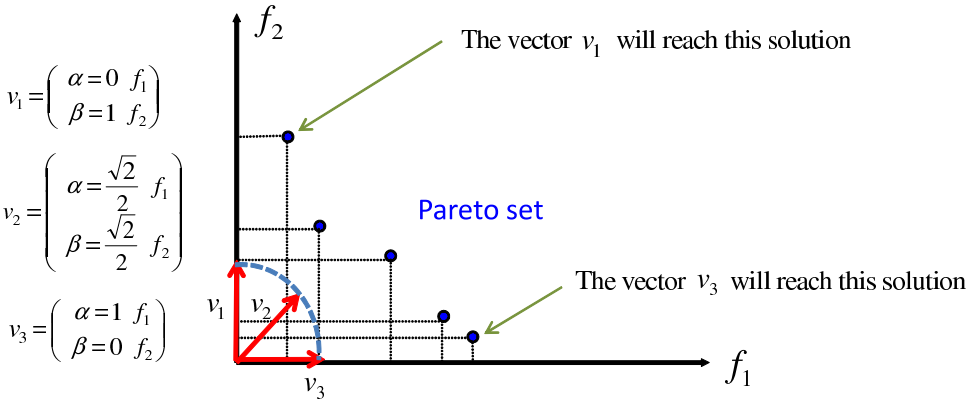



Figure 2.6: The vector meta-selection mechanism applied to a bi-objective Pareto set in order to choose a particular solution.

both the Pareto approach with several non-dominated solutions and its adequate mechanism for the provider to fit at best the SLA. In addition, the stored results obtained using MO-GA in the Pareto archive do not give the possibility for the algorithm to start a new process of scheduling because of several equivalent potential solutions in the Pareto set. 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 federation 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 objective(s) to promote. And the last one, our solution, that uses a vector as an input parameter in order to automate the progression of the experimentations. Our vector parame-

ter 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 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 Figure 2.6, we give an example with three two-dimensional vectors. In Figure 2.7, we give an example of transition from an old state to a new one. The example concerns a four processors cloud within a cloud federation where the applications are represented by $Ai$ and the processors by $Pj$.

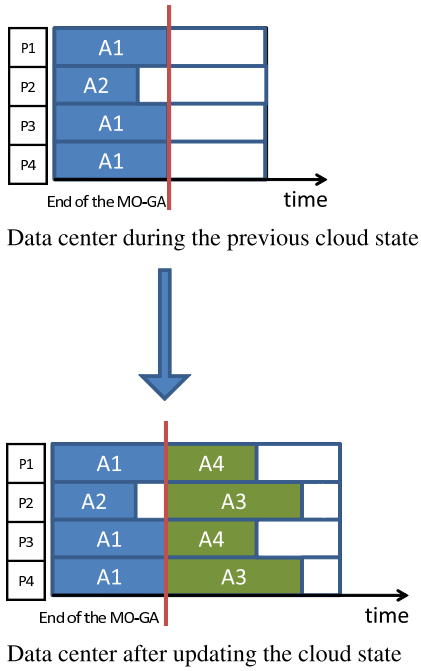



Figure 2.7: The cloud state transition within a federation cloud after the end of the MO-GA execution.

## 2.4 Experimental Evaluation

This section presents the results obtained from our comparative experimental study of the proposed Meta-Scheduler of the Cloud Federation (MSCF). The experiments aim to demonstrate and evaluate the performance of the multi-objective evolutionary algorithm using different meta-selection policies. It also aims to compare the obtained results of MSCF algorithm to a maximum application consolidation-based scheduling heuristic and to a random approach.

### 2.4.1 Experimental settings

As for the modeling section, the experimental settings concern both side of our model, client side with its HPC applications and provider side with the hardware configuration of the cloud federation.

- **Application settings:** Since our approach deals with HPC applications, we use realistic workloads traces from Feitelson's Parallel Workload Archive (PWA) [Feitelson 2009]. 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. Furthermore, the reason why we choose the period between June 2010 and August 2010 in the RICC instance is the high utilization rates and the pick of load that offers this period. The information that we extract from both instances are the submission 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 [Venugopal 2008] 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 get 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 both 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 characteristics as in [Garg 2010] (see Table 2.1). The $COP$ value of each cloud is given by a uniform distribution between [0.6, 3.5] as indicated in [Greenberg 2006]. The electricity prices and carbon emission rates are taken from respectively US Energy Information Administration (EIA) report [EIA 2007] and US Department of Energy (DOE) [DOE 2007]. Since we are dealing with a meta-scheduler, we do not use energy reduction techniques within the clouds. Hence, the optimal frequencies of the processors in the clouds are not used.

Table 2.1: Characteristics of the clouds which compose the cloud federation.

| Location | COP rate | $CO_2$ rate (kg/kW h) | Electricity price ($/kW h) | CPU: $\alpha$ | CPU: $\beta$ | 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 |

### 2.4.2 MSCF 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 2.3.6 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 favor to the profit criterion. Regarding the arrival rate variation, we vary the original workload by changing in each arrival rate the submission time of the HPC 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 one switches from an arrival rate to another one divides the submission time by 10. Thus, by shortening the submission time of the applications we increase the workload. The client price is fixed as the twice of the average electricity cost of the clouds in the federation. Scheduling cycle in our algorithm is set to 50s. Table 2.2 summarizes the parameters used in our experiments.

Table 2.2: 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 |

#### 2.4.2.1 MO-GA algorithm parameters

As previously mentioned, MSCF algorithm is based on a multi-objective genetic algorithm MO-GA to find the assignments of the services over the cloud federation. Table 2.3 summarizes the different configuration parameters of MO-GA.

Table 2.3: MO-GA parameters.

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

### 2.4.3 Maximum applications consolidation-based scheduling heuristic and random approach

To the best of our knowledge, there are 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 a 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) [Johnson 1973]. After the *scheduling cycle* and the arrival of a new pool, the heuristic aims to maximize the QoS of the client (the number of correctly assigned applications). To do so, it chooses randomly a cloud among the federation and fills it by the maximum number of requests by consolidating the applications. When the cloud cannot support the application requirements the heuristic chooses another cloud and so on until it finds a cloud which satisfies the requirements and respects all the constrains. 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 clouds.

The random approach is based as its name indicates on a random assignment of the HPC applications to the clouds that compose the federation according to the arrival rates. The obtained assignments after each *scheduling cycle* are evaluated in a multi-objective way, according to both the number of successfully scheduled applications and the value that these assignments obtain regarding the three addressed objectives. The final result for each instance over the whole workload is the sum of the fitness of each objective of the obtained results during each *scheduling cycle*.

### 2.4.4 Performance evaluation

This section depicts a bench of experiments with different parameters in order to validate the proposed MSCF algorithm. As said previously, in addition to optimizing the three objectives, the aim of MSCF is first to satisfy the maximum number of clients QoS. In other words, the MSCF has to successfully assign the maximum number of HPC applications. A comparison between our approach, a maximum applications scheduling consolidation-based heuristic (Heuristic) and a random assignment-based approach (Random), both presented in Section 2.4.3, seems to be the best choice to evaluate our work.

In order to switch from a cloud state to another we used 4 different vectors (see Table 2.2). These vectors help through their coordinates to choose the type of the solution (scheduling) that will be used for the cloud transition 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 2.4 to 2.11.

The results for each instance (LLNL and RICC), for each arrival rate and for each meta-selection vector configuration of the MO-GA have been deducted 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 aforementioned algorithms are deduced also respectively from 80 and 30 independent runs. Note that, the random part of the heuristic concerns only the cloud selection phase. In addition, the depicted values in the presented results are the medians of the samples results. Indeed, because of the non-normality of the distributions of all 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 consolidation-based heuristic are presented in the Tables 2.14 and 2.15.

First, the 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 broad interval of values on all the objectives (see Table 2.10 and Table 2.11). We deduce that this vector setting helps the provider to control the progression of the results over the different arrival rates. Besides, we also notice that the more the application rate is high less efficient are the results and the higher the number of failed applications is. Moreover, since this vector policy does not favor any objective, we obtain less sharp results when compared to ones obtained by other vector orientations. On the other hand, the vector orientations that favor a specific objective allow one to obtain a significant improvement on this specific objective. In other words, compared to the other meta-selection policies (vectors), the improvement of the objective which is concerned by the policy is significant compared to the other policies with a different orientation. Moreover, this improvement of solution quality concerns more the *Low* and *Medium*

Table 2.4: Experimental comparison for the LLNL Thunder instance, between the MSCF 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 |

arrival rates than the *High* and *Very high* 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 when favoring a given objective. The best value for a given objective for those kind of arrival rates is obtained with other orientation vector. This can be explained by the local optima phenomenon. 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 conduct the future incoming applications to be assigned on clouds with worse characteristics regarding the considered objective, which leads to uninteresting results on the objective itself. One can see an illustration of this observation in the instance LLNL for the energy-oriented vector table Table 2.4. We obtain in that table for a *Very high* arrival rate better $CO_2$ emissions than in Table 2.8 where the vector is favoring the $CO_2$ criterion.

We notice this behavior more often in the experiments using the RICC instance.

Table 2.5: Experimental comparison for the RICC instance, between the MSCF 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 |

This is caused by a higher utilization rate proposed by the interval of the RICC instance on which we conduct our experiments. Indeed, both RICC and LLNL instances have about the same number of applications while the time interval of the LLNL instance is longer (6 months) than the one of the RICC instance (2 months). Therefore, in Table 2.7 which favors the profit, the $CO_2$ emissions are lower than in Table 2.9 which favors $CO_2$, always regarding the very high arrival rates. The same behavior is noticed in Table 2.7 which is profit-oriented, where the earned profit is lower for the *Very high* arrival rates than in Table 2.9 which favors $CO_2$. This is due as previously explained for the LLNL instance, to the fact that changing the orientation of the vector helps the algorithm to escape 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 the 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*

Table 2.6: Experimental comparison for the LLNL Thunder instance, between the MSCF 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 |

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 to maximize the QoS of the clients.

To be as fair as possible and do not favor any of the criteria, the comparison of the heuristic with MSCF was done with an *Average* orientation vector policy. 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

Table 2.7: Experimental comparison for the RICC instance, between the MSCF 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 |

instance like LLNL. The details of the comparison study between our algorithm and the heuristic for each instance and for each arrival rate are presented in Table 2.14 and Table 2.15. Indeed, the values in Table 2.14 for the LLNL Thunder instance show an improvement of the results obtained by MSCF compared to the maximum applications scheduling heuristic for all the arrival rates. However, the improvement decreases according to the increase of the arrival rate. The best improvement of 51% concerns the $CO_2$ emission reduction for the *Low* arrival rate. Regarding the RICC instance results in Table 2.15, the improvement concerns all the arrival rates except the *Very high* arrival rate. Indeed, this deterioration in the results is explained by the local optima phenomenon. When a high rate of application arrives, the MO-GA tends to optimize the criteria for only the current arrival of applications regardless of the next possible arrivals. In fact, because of the real time arrival, MSCF ignores the existence of other application arrivals. On the other side, the heuristic which does not saturate the interesting resources, because of a less optimal solution, can

Table 2.8: Experimental comparison for the LLNL Thunder instance, between the MSCF 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 |

benefit from those resources later during the next arrivals of applications. It obtains therefore, better final results. The same explanation accounts 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.

Regarding the time consumption of MO-GA, the results show that the heuristic gives results faster than MO-GA. However, this difference in the computation time does not impact 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 MSCF 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. One can deduce then computing 6 months of applications in less than

Table 2.9: Experimental comparison for the RICC instance, between the MSCF 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 |

a day means that the MO-GA's processing time at each application arrival is easily covered by the *scheduling cycle* time. In other words, each pool of applications is scheduled in less than 50 seconds.

Finally, 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 cannot handle the constraints. Therefore, it 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, for the instance LLNL as well as for RICC (see random part in Table 2.4 and Table 2.5).

Table 2.10: Experimental comparison for the LLNL Thunder instance, between the MSCF 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 2.11: Experimental comparison for the RICC instance, between the MSCF 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 2.12: Comparison of the number of failed applications on the LLNL Thunder instance between the MSCF 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 2.13: Comparison of the number of failed applications on the RICC instance between the MSCF 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 2.14: Percentage change between the MSCF algorithm using an average orientation vector and the heuristic on the LLNL Thunder instance, according to the different application arrival rates.

| | Comparison according to criterion (MSCF algorithm 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 2.15: Percentage change between the MSCF algorithm using an average orientation vector and the heuristic on the RICC instance, according to the different application arrival rates.

| | Comparison according to criterion (MSCF algorithm 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% |

## 2.5 Conclusion

In this chapter, we presented an efficient meta-scheduler (MSCF) using a multi-objective genetic algorithm (MO-GA) designed for service-level scheduling (HPC service). The challenges in this work were to minimize the energy consumption, the gas emission and maximize the profit of a geographically distributed cloud federation while respecting the SLA with the client. The main contributions of this chapter are the following:

- *Meta-scheduler:* we showed that high-level scheduling (service-level) has interesting impacts dealing with important issues such as energy consumption, profit or QoS. Indeed, a meta-scheduler deals with large geographical infrastructures, usually deployed over several sites. This geographical distribution offers the meta-scheduler a high heterogeneity regarding different features over the infrastructure, which opens a lot of optimization possibilities mainly using scheduling.

- *Multi-objective approach:* the federation cloud raises several issues formulated as conflicting criteria. Therefore, since each criterion impacts the others, we proposed a multi-objective genetic algorithm (MO-GA) that considers all the criteria at once to have a Pareto optimization that does not neglect any of them. As a metaheuristic, MO-GA successfully raised this challenge by providing a wide range of solutions proven as efficient.

- *Meta-selection vector:* one knows that the market-oriented model in the service-level scheduling requires compromises. Therefore, the Pareto set of solutions proposed by the MO-GA in the MSCF gives the advantage to offer plenty of equivalent solutions regarding different criteria. However, to benefit from this diversity, a selection mechanism is needed. In this purpose, we proposed such mechanism as a vector form, called meta-selection, to help the provider to achieve at any time the QoS of the client by acting directly over his/her criteria according to the needs.

Chapter 3

# Task-level Scheduling in a Cloud Brokering Environment

### Main publications related to this chapter

- Y. Kessaci, N. Melab and E-G. Talbi, A Pareto-based Genetic Algorithm for Optimized assignment of VM Requests on a Cloud Brokering Environment, *Internationnal IEEE Congress on Evolutionary Computation*, June 20-23 Cancun, Mexico 2013.

### Contents

## 3.1 Introduction

Dealing with cloud computing issues from a high level point of view by applying service-level scheduling offers advantages. However, these advantages reach their limit and need a complementary level of scheduling once dealing with the proper scheduling inside each cloud of the federation. Indeed, this granularity does not allow to address all the needs of the market-oriented cloud paradigm, mainly the finer needs of the client like defining the tradeoff between the price and the performance of his/her task.

Therefore, the assignments resulting from a service-level scheduling often do not fit the exact needs of the client. In that case, the client faces the issue of the high variability and diversity of QoS proposed by the cloud to which his/her service is assigned. This diversity is due to the difference in the features of the instances proposed by the provider. The client faced to such large possibilities, gets unable to know or to choose the good instances at the right moment for his/her task. Hence, one can see the interest of having a lower level of scheduling (task-level scheduling) to complete the service-level scheduling. In this case, a new tier (broker) is required between the provider and the client, to dynamically find a tradeoff between the two parts. The broker has to follow a utility model to match the provided resources to the clients' requirements.

Many issues from the survey presented in Section 1.4.2.1 arise for task-level cloud scheduling. To address all those issues, the contribution of this chapter differs from the previous studies in plenty aspects. We use the concept of utility computing and present a new approach that depicts and focuses on the two main criteria that affect task-level scheduling: the price of the Virtual Machines (VM) instances and their response time when executing the client's tasks. The aim is to dynamically minimize both the criteria in order to give the best quality of service (QoS) to the clients while providing an interesting profit for the cloud broker. In that purpose, we propose a multi-objective genetic algorithm for cloud brokering (MOGA-CB). It provides a set of Pareto optimal assignments by dispatching the client's tasks over the best combination of VM instances with the minimum cost and response time. The experimental validation of MOGA-CB uses real information provided by the infrastructure service provider (e.g. Amazon) to retrieve the prices of the instances and their different performances.

The remainder of this chapter is organized as follows. Section 3.2 presents the system, satisfaction and profit models used in our problem modeling. The proposed algorithm is presented in Section 3.3. The results of our experimental study are discussed in Section 3.4. Finally, the conclusion is drawn in Section 3.5.

## 3.2 Task-level Scheduling Model

### 3.2.1 Brokering model

In this chapter, the focus is made on the scheduling within a single cloud to find the best task assignment on a set of proposed VM instances. This is carried out by an intermediate called the broker. Moreover, an IAAS cloud type is considered in this work. Therefore, due to the aforementioned additional tier, the model evolves to a three-tier architecture where the tiers represent the clients, the cloud provider and the cloud broker (see Figure 3.1). As said previously, the role of the broker is to find the best configuration among the resources proposed by the provider to fit the client tasks needs. Besides, since dealing with a market-oriented cloud, the broker charges the client for the provided service which is in this case a lower level service. Thus, unlike the infrastructure service studied in Chapter 2, which is directly related to the provider of the cloud federation, the service addressed in this chapter is the one provided by the broker. Note that to distinguish between the service of the broker and the service of the provider, we name the former the business service while we call the latter the infrastructure service.

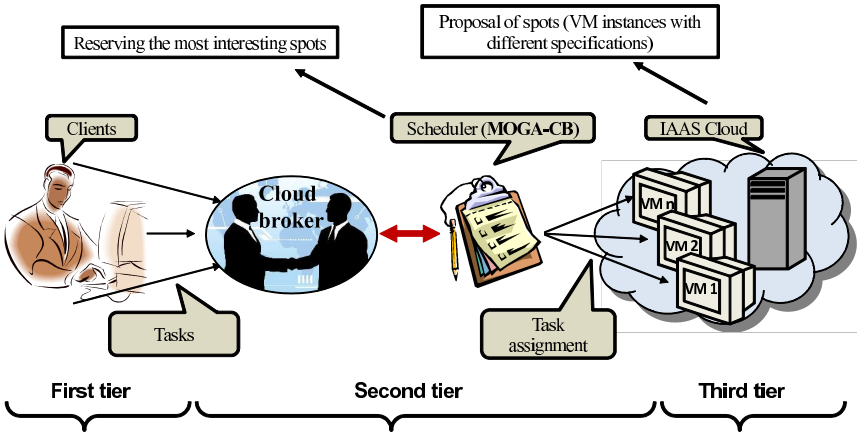



Figure 3.1: A representation of our three-tier cloud model.

The market changes impact the prices of the instances according to offers and demands. The VM instances are proposed for rental by the infrastructure service provider. The broker reserves those instances according to the demands of the clients. Depending on the offers over the VM instances the prices may change. The variation of the prices of the VM instances is bounded by a time duration called *scheduling round* after which the instance's price does not change. As for the major market-oriented cloud models the reservation time of the resources is divided into slots. Hence, the user has to pay for the whole time of the reserved slot even if his/her effective time usage of the resources is less than the time slot. This is inspired from the Amazon EC2 Spot [Ama 2013].

Each client's task request is a quadruplet ($size$, $S_{rate}$, $\alpha$, $\beta$), where $size$ represents the size of the task in terms of CPU computation time and $S_{rate}$ is the satisfaction

level of the client. In other words, it represents the roughness of the client (i.e. a high value means a demanding client while a small value is for a non demanding one). This value is a rate over the best solution provided by the broker. Finally, $\alpha$ and $\beta$ designate the preference that the client has for respectively the service price and the response time.

The originality of this approach is to propose a Pareto scheduling approach using a multi-objective genetic algorithm in order to allow the broker to dynamically find the best choices of VM instances among the ones proposed by the infrastructure service provider. The objectives considered in our algorithm are the price and response time of the VM instances while executing the tasks of the clients. Those two objectives are the main criteria for expressing the client's satisfaction and to deduce the broker's profit. In other words, our algorithm aims to provide the best Quality of Service (QoS) by satisfying as much as possible the client, while allowing the broker to maximize its profit. The optimization of the objectives is based on the diversity of the tasks characteristics of the clients and both the fluctuations of the prices and the difference in the performance of the VM instances.

### 3.2.2 Satisfaction and profit models

In Section 1.4.2.1, we have shown that several state-of-the-art works deal with QoS focusing only on one criterion such as the response time or the amount of satisfied requests. The work in [Chen 2011] based on the utility theory in economics [Mankiw 2008], proposes a client's satisfaction modeling. As defined in Equation (3.1), this client satisfaction is based on the service price $p$ and the response time $t$.

$$Satisfaction(p,t) = S_{max} - \alpha p - \beta t \tag{3.1}$$

where $S_{max}$ is the maximum satisfaction value that the broker can deliver to the client. The $\alpha$ and $\beta$ parameters are used to give more significance to one of the criteria (the price or the response time) without changing the client satisfaction. The ratio $\alpha/\beta$ or $\beta/\alpha$ is known as marginal rate of substitution in economics. Since our work is based on a Pareto approach, the $(\alpha, \beta)$ parameters allow one only to make a choice of the optimal solution (i.e. that fits better the client's needs). We will also prove latter that using the Pareto approach we can simplify Equation (3.1) by removing the $\alpha$ and $\beta$ values from the client's request parameters without changing the client's satisfaction.

In addition, Equation (3.2) is deduced from Equation (3.1) in order to compute the broker's profit. Indeed, with the parameter $S_{rate}$ presented previously we can deduce the satisfaction needed by the client. Therefore, knowing the response time of the chosen VM instance, the $(\alpha, \beta)$ parameters and the maximum satisfaction $S_{max}$ proposed by the broker one can deduce the profit generated by the broker while providing its business service to the client.

$$profit = (S_{max} - Satisfaction \times S_{rate} - \beta t)/\alpha \tag{3.2}$$

### 3.2.3 Broker-based task-level scheduling modeling

In the task-level scheduling model, the role of the broker is to find the best assignment of the tasks by choosing the right combination of the proposed VM instances based on their current price, their performance index $PI$ and the load in terms of client's needs. The performance index $PI$ is a normalization value that helps to compare the performances of the different VM instances when computing the same task. The different values of $PI$ of the used instances in this work are presented in Section 3.4. The problem consists then to schedule $J$ client tasks on $N$ VM instance types. Moreover, we know that the task scheduling problem is generally NP-hard [Garey 1979]. Therefore, as for the service-level scheduling the task-level scheduling on a brokering environment is NP-hard as well. Besides, unlike the static service-level scheduling, to better meet the clients' needs, the task-level scheduling is dynamic. Thus, addressing this problem justifies the use of metaheuristics such as the proposed MOGA-CB genetic algorithm.

In our model, the client submits requests to execute his/her tasks with QoS requirements. Those requests are defined by two types of parameters. The first ones are fixed by the client himself/herself at the submission of the request, while the others are used by the algorithm and deduced *a posteriori*. The requirements of the user as said previously, are the request size, the satisfaction rate and the $\alpha$, $\beta$ values designated for a task $j$ by the tuple $(size_j, S_{rate,j}, \alpha, \beta)$. The other types of parameters defined during the scheduling process, serve to inform about the state of the task during the process. The first variable $cost_{ji}$ represents the cost of the task $j$ when assigned to the VM instance $i$ with its current price during the scheduling round duration (i.e. no price fluctuation during each scheduling round). The other variable $rpt_j$ represents the remaining processing time of the task $j$ on a standard instance. The initial value of this variable is the parameter $size_j$. The $rpt_j$ value over a specific instance $i$ noted $rpt_{ji}$ is given by Equation (3.3), where $PI_i$ is the performance index of the instance $i$. This value is updated only in case where the task is moved and assigned to another instance before the end of its execution.

$$rpt_{ji} = rpt_j / PI_i \tag{3.3}$$

The objective functions of our approach aim to **minimize** the total task cost designated by $MinCost$ and the response time of the tasks designated by $MinRT$ in order to provide the best client satisfaction and broker profit. It is formulated as follows:

$$MinCost = \sum_{j}^{J} \sum_{i}^{N} cost_j + rpt_{ji} \times price_i = \sum_{j}^{J} \sum_{i}^{N} cost_{ji} \tag{3.4}$$

where, $cost_j$ is the previous cumulated cost of the task $j$, $rpt_{ji}$ is the remaining processing time of the task $j$ if processed on the instance $i$, and $price_i$ is the instance price during the current scheduling round of the instance $i$.

$$MinRT = \sum_{j}^{J} \sum_{i}^{N} currentTime - arrivalTime_j + rpt_{ji} \tag{3.5}$$

where, $currentTime$ is the current time, $arrivalTime_j$ is the arrival time of the task $j$ and $rpt_{ji}$, as in Equation (3.4), is the remaining processing time of the task $j$ on the instance $i$, if not concerned by a reassignment.

## 3.3 The Proposed Multi-objective Genetic Algorithm for Task-level Scheduling

In this section, we describe in details the steps of our scheduler together with the multi-objective genetic algorithm for cloud brokering. Note that in the following we designate by *MOGA-CB scheduler or MOGA-CB* the whole steps of the scheduling process (i.e. the scheduler), while the notation *MOGA-CB algorithm* represents only one step of the *MOGA-CB scheduler* which is the Pareto multi-objective genetic algorithm step.

### 3.3.1 MOGA-CB scheduler steps

The MOGA-CB approach is a metaheuristic based scheduler. The aim of this algorithm is to be used by the broker to reach its objectives. Before each scheduling, the MOGA-CB scheduler waits as previously said for a fixed period of time called scheduling round. This period allows one to gather a pool of tasks in order to have a larger choice in the assignment and thus to optimize the future scheduling. In addition, the scheduling round duration is used by the algorithm to retrieve the current instances' prices.

Once this phase passed, the pool of tasks is managed by the MOGA-CB algorithm to find the best possible assignments over the different VM instances. The result of the execution is stored in a Pareto archive. Once the set of Pareto solutions (assignments) is proposed, the algorithm chooses one scheduling among this set according to the client's settings to fit at best his/her satisfaction. The selected solution from the Pareto set is also used as a state to update the algorithm parameters and to inject back the not yet finished tasks to follow their execution during the next scheduling round. This last step helps to get a basis from which the next iteration will start to handle the next pool of tasks. The algorithm will perform another round including in its scheduling both the pool of not finished tasks and the pool of newly arrived tasks. The scheduler keeps iterating until no more tasks arrive and no more tasks are still being processed. All the scheduling steps are drawn in Figure 3.2.

### 3.3.2 Task-level scheduling encoding

As presented in Section 1.3.2.1, the solution representation is a very important step in the design of a metaheuristic. We propose as encoding for the MOGA-CB

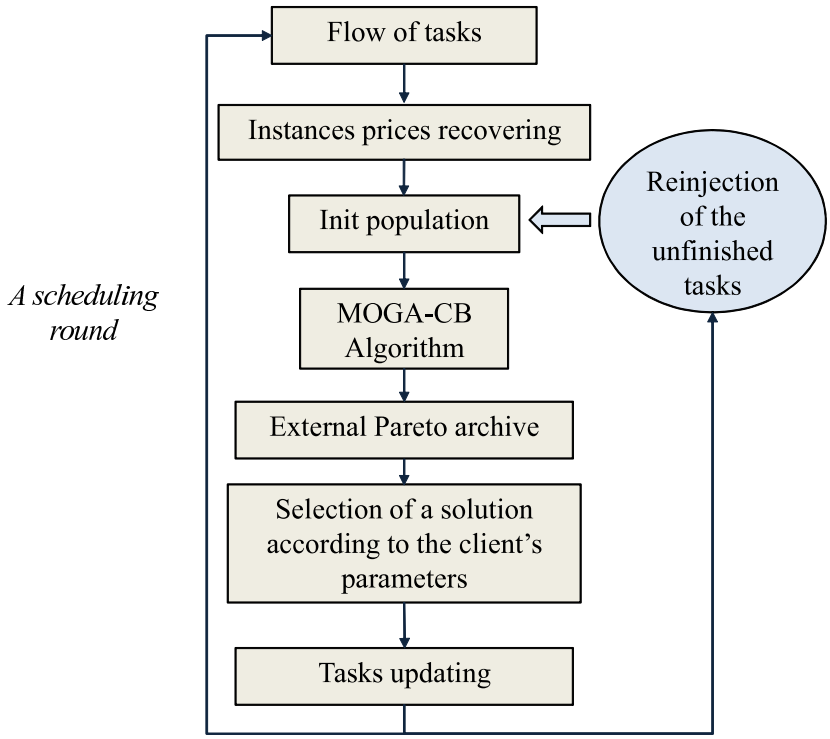



Figure 3.2: The Flowchart of the MOGA-CB scheduler.

solutions the vector illustrated in Figure 3.3.

The encoding is close to the one presented for addressing the service-level scheduling problem. However, some differences remain regarding the meaning of the values. Figure 3.3 represents one possible scheduling among plenty provided by the MOGA-CB algorithm. In the proposed example we identify two major information. The indexes of the vector depict the tasks that are scheduled and the number in each cell identifies the VM instance to which the task is allocated. In other words, the first cell represents the first task of the pool that is currently handled by MOGA-CB algorithm. In this case, this task is allocated to the VM instance 2. The second task is allocated to the VM instance 0, and so on. This encoding informs about the number of tasks currently addressed and that have *rpt* value different from 0 (i.e. not finished yet tasks), which is 10 in our example. This encoding allows one also to deal with the characteristics of our problem. Indeed, it enables one to schedule all the tasks of the pool by assigning each of them to only one VM instance at a time. Nevertheless, a VM instance can be chosen for more than one task. Note that not all the VM instances are necessarily used in each solution and that a task can move from a VM instance to another before it is finished. We also assume that the broker can always ask the infrastructure provider for new available VM instances. This is realistic since our approach can be used for a hybrid cloud with several providers. We can afford then to do not deal with the availability of the VM instances.

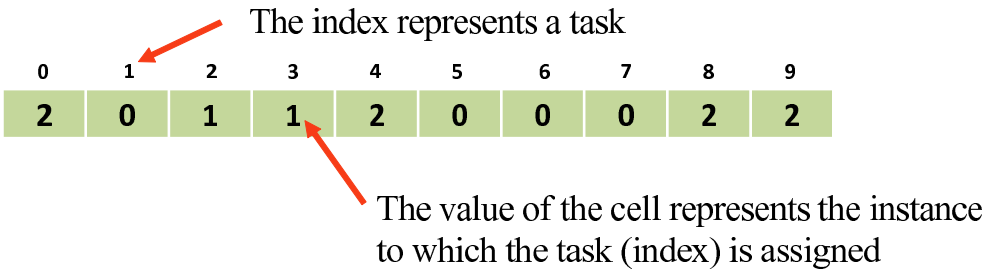



Figure 3.3: Task-level problem encoding.

### 3.3.3 Population initialization

This step affects directly the quality of the future results of a metaheuristic. Thus, as dealing with a genetic algorithm, taking care of this phase is really important for the quality of the future results. In our approach, the initialization of the population is done in two steps as follows:

- The first step initializes each individual by the not yet finished tasks.
- The second step initializes the rest of the solution using a random method.

Therefore, at the first step the algorithm checks the number of not yet finished tasks from the previous scheduling round. After that, it adds them to the initial solution by assigning them to their previous instances to let them resume their processing. The second step starts only when the first step is finished or when the first step is not necessary. This can happen for example during the first iteration of the algorithm where no request from the previous scheduling round is being processed. The role of the second step consists of initializing the rest of the chromosome (solution) by assigning the other new arrived tasks randomly to the VM instances. Note that not all the individuals of the population are assigned following both steps. Indeed, to add diversity, the initialization of some individuals is done following only the random method even regarding the assignment of not yet finished tasks of the previous scheduling round.

### 3.3.4 Multi-objective genetic algorithm MOGA-CB

In order to find the best assignments and to schedule all the arriving tasks, the MOGA-CB algorithm performs at each scheduling round as follows: It starts by using the initialized population (as presented in Section 3.3.3) to generate new assignments (offsprings) with the mutation and crossover operators. Each modification, caused by the variation operators, consists to explore a new assignment for the different tasks of the scheduled pool. Each modified individual (solution) is subject to the application of an evaluation operator (fitness) that evaluates the resulting offsprings. This fitness is deduced from both the total price of the tasks and their total response time for each scheduling (solution). The next step of the MOGA-CB

**Algorithm 6** Pareto MOGA-CB algorithm.

1: **Input:** An updated set of VM instances and a set of new and old tasks
2: InitPopulation(VM instances, Tasks, pop);
3: Fitness(pop);
4: DominanceRanking(pop);
5: Calculate the crowding distance;
6: Selection(pop, parent1, parent2) ;
7: Crossover(parent1, parent2, offspring1, offspring2) and/or Mutation(parent1);
8: Replace(offspring1, offspring2, pop);
9: **Output:** A set of Pareto non-dominated solutions (assignments)

algorithm is first, the selection of the best solutions among the population, then the replacement of other solutions according to a replacement strategy. In this purpose, the number of individuals in the population is kept constant. The selection operator of our genetic algorithm is based on a tournament strategy. Performing that way leads the MOGA-CB algorithm to explore a wide range of possible assignments and keeping the best ones through the different rounds. The algorithm stops when no improvement on the best solution is observed after a fixed number of generations. We note also that both mutation and crossover, help the MOGA-CB algorithm at each new scheduling round to dynamically modify the assignment of previous not yet finished tasks to explore more possibilities. Indeed, the change brought by the new arriving tasks at each scheduling round changes the solutions' landscape. The pseudo-code of MOGA-CB algorithm is presented in Algorithm 6. Its evolutionary core is based on the NSGAII [Deb 2002] multi-objective genetic algorithm. Thus, the method used to rank the individuals of the population, because of the multi-objective context, is the dominance depth fitness assignment. This archive contains the different non-dominated solutions progressively generated. Besides, the selection process of our genetic algorithm is based on two major mechanisms: elitism and crowding. They allow one respectively the convergence of the evolution process to the best Pareto front and maintaining some diversity of the potential solutions. More details about these techniques are provided in [Talbi 2009]. The principle of our mutation and crossover operators are the same as presented in Section 2.3.4.

### 3.3.5 Satisfaction-based selection mechanism

By using the standard economic formula given by Equation (3.1), the obtained satisfaction results can be drawn as parallel lines (see Figure 3.4). In other words, each line represents a given satisfaction $S_i$. Sliding toward each axis in Figure 3.4 according to both $\alpha$ and $\beta$ parameters gives more importance to one of the objective holding the same satisfaction. A closer point to the response time axis gives better response time values while a closer point to the price axis gives better pricing results. To increase the client's satisfaction, the sliding has to be done for both objectives at the same time toward the origin of both axis (i.e. price=0 and response time=0).

Moreover, it is obvious that the client seeks for this type of improvement at first to increase his/her satisfaction. However, the method offers to the client the ability to make only a choice within the same satisfaction, giving each time more importance to one of the objectives then both objectives at once. Therefore, a Pareto approach appears to be the best choice to provide the best satisfaction at each time regardless the parameters. Indeed, a Pareto approach natively provides the set of solutions that is the closest to the origin point of the axis by giving non-dominated solutions (see Figure 3.5). In other words, this method returns the best satisfactions and the ability to move between Pareto equivalent satisfaction solutions using both the $\alpha$ and $\beta$ parameters only to choose a solution. In this purpose, we propose a mechanism that deals with the set of Pareto solutions obtained by the MOGA-CB algorithm to provide at each time the best satisfaction to the client. Moreover, our method helps also the task scheduler to switch from processing a pool of tasks to a new one by selecting a single solution among all the proposed solutions. This selection step comes right after the end of the processing of the genetic algorithm for each scheduling round. It aims to pick up a solution among the Pareto set according to the client needs, in order to update the variables of the remaining processing tasks from the last scheduling round. This state will be the starting point from which the next execution of the MOGA-CB scheduler will assign a new pool of tasks. The idea behind choosing a Pareto approach in our work is to propose to the broker as more non-dominated solutions as possible. Each one of these solutions is better than the others regarding a given objective. Unlike using Equation (3.1) to find the assignments, the satisfaction-based selection mechanism uses only this equation to select a solution according the user's preference. A solution in our algorithm is defined by its price $p$ and response time $t$. All the solutions that are chosen from the Pareto set are always the ones that give the best satisfaction value. However, based on $\alpha$ and $\beta$ parameters for the same satisfaction, the user can choose a solution according to his/her needs.

Furthermore, the MOGA-CB scheduler proposes a higher level parameter to promote either the broker's profit or the client's satisfaction. Indeed, with that additional parameter, the algorithm can even satisfy the broker requirements by providing him/her for example a certain benefit rate based on the Pareto optimum solution cost, or increases the client satisfaction by providing him/her the requested satisfaction rate. The default setting of this parameter always satisfies the clients and provides good solutions in terms of broker's profit. The only interest of disabling the default setting parameter lies for instance in the case where the broker is in deficit. The profit option should be temporarily chosen despite the unsatisfied client to recover a correct budget level. This parameter allows to give an orientation to the Pareto front. It can use the profit and response time either, to draw by default a client oriented Pareto front with as optimized value the satisfaction, or to draw a Pareto front with the aim to increase the profit of the broker.

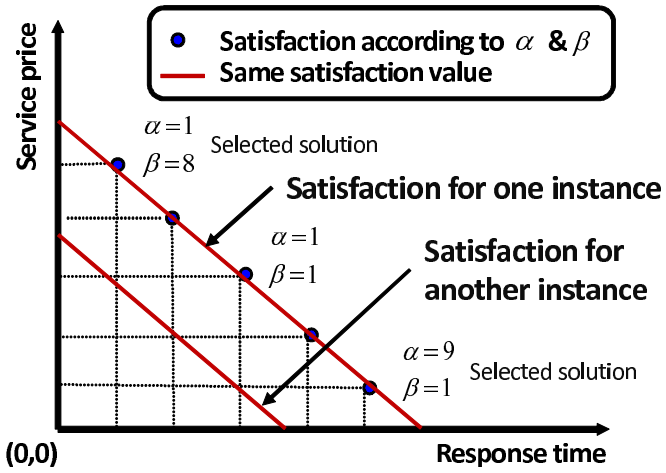



Figure 3.4: An example of different levels of satisfaction using an aggregation method.

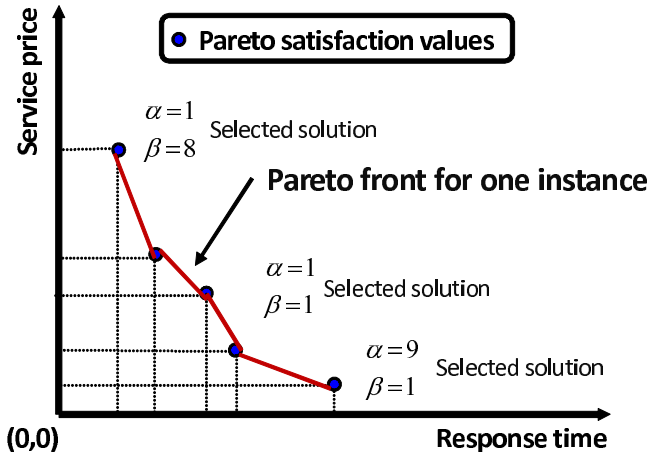



Figure 3.5: A set of non-dominated satisfactions using a Pareto method.

## 3.4 Experimental Study

This section presents the results obtained from our experimental study. The experiments aim to demonstrate and evaluate the contribution of the multi-objective evolutionary approach over the $\alpha$, $\beta$ tradeoff and the different behaviors that can have the MOGA-CB scheduler according to the variations of the input parameters, the performance indexes and the prices of the instances.

### 3.4.1 Experimental settings

The experimental settings concern both the client and the infrastructure provider sides of our three-tier model: the client side with the tasks and the provider side with the VM instances.

- **Tasks' settings:** regarding the inputs of the broker's scheduler, we generated tasks arrivals according to a Poisson process. Each task arrives at a given slot of the scheduling round to which it belongs. The scheduling round slot equals 1/10 of the scheduling round. A task's starting time equals to its arrival time (i.e. it is the time value in the scheduling round to which it belongs) plus the waited slots in this scheduling round (see Figure 3.6). Moreover, the task information in our experiments vary according to four parameters. Indeed, as said in Section 3.2.3 with the tuple $(size_j, S_{rate,j}, \alpha, \beta)$, the tasks differ by their size (execution time), their client satisfaction rate and finally the $(\alpha, \beta)$ parameters. Therefore, we generated the elements of this quadruplet, where the execution time $size_j$ is a value from [2, 50] which represents the duration of the request in terms of number of scheduling rounds. The client satisfaction rate $S_{rate,j}$ varies in the interval [0.1, 1] where on the one hand, the value 0.1 represents the less demanding client (10% of the best obtained satisfaction is enough to gratify him/her) and on the other hand the value 1 is the most demanding client (he/she needs 100% of the best obtained solution). The $\alpha$ parameter varies in the set $\{9, 3, 2, 1\}$ and $\beta$ in $\{8, 4, 2, 1\}$.

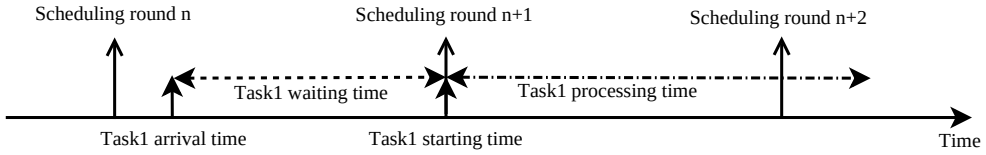



Figure 3.6: The task starting processing time *vs* the task arrival time in the scheduling round.

- **VM Instances settings:** for the provided infrastructure service instances, we used the instances proposed by Amazon EC2 [Ama 2013]. We used three types of instances: the *small* one, the *large* one and *extra large* one. We deduced the performance $PI$ indexes of these instances from the work proposed

in [Chen 2011]. All the parameters' values are reported in Table 3.1. Regarding the fluctuation of the prices of instances, we downloaded them from the Amazon pricing history [Ama 2012a]. The price fluctuations of all the previously cited types of instances extend over a period of one month on the US California site. They are drawn in Figures 3.7, 3.8 and 3.9.

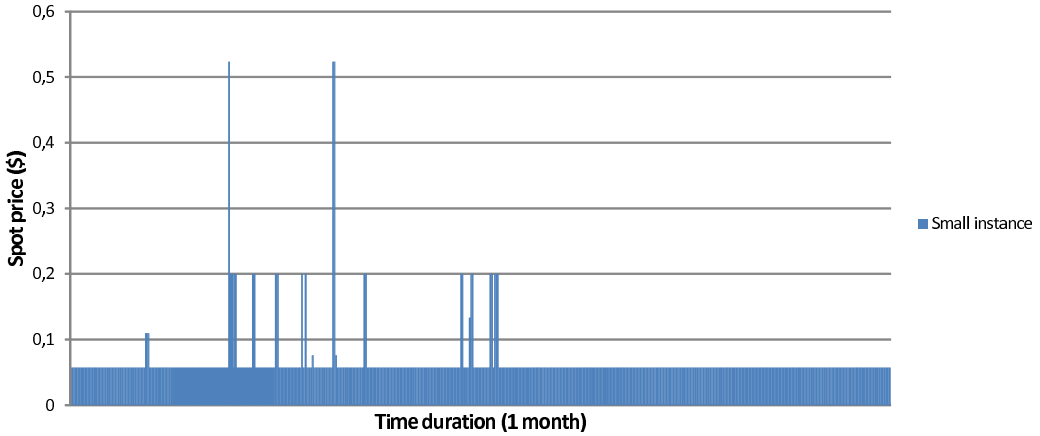



Figure 3.7: The price fluctuation of an Amazon's small instance over a period of one month.

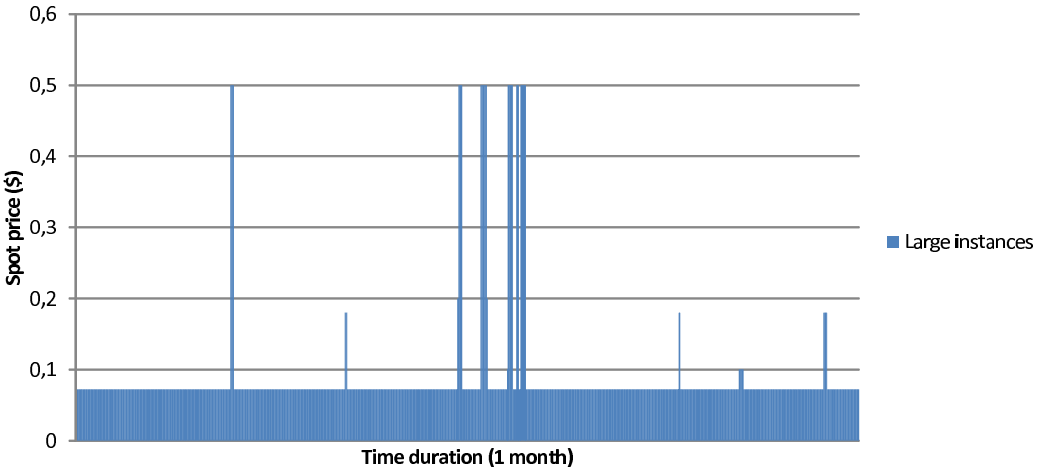



Figure 3.8: The price fluctuation of an Amazon's large instance over a period of one month.

### 3.4.2 MOGA-CB scheduler parameters

In our experiments, we used some parameters such as the satisfaction/profit selection parameter, the arrival rate of the tasks and the task execution time. The satisfaction/profit selection parameter is used in order to promote at a certain time

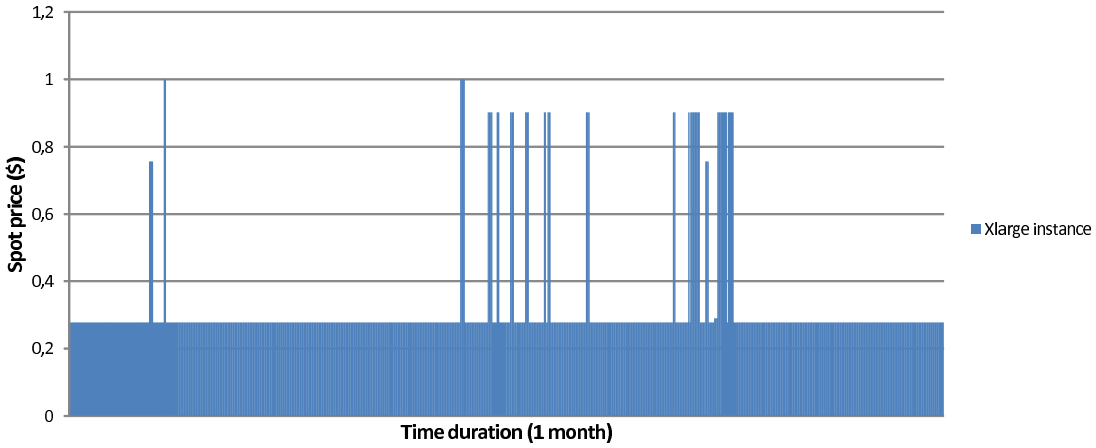



Figure 3.9: The price fluctuation of an Amazon's extra large instance over a period of one month.

the profit of the broker in case of budget difficulties. Once this option activated, another parameter is added to the algorithm to inform it about the profit rate needed by the broker. We performed experiments with both state options (enabled and disabled). Regarding the variation of the task arrival rate we used a Poisson process with a $\lambda$ rate of 15 per scheduling round. The number of tasks is 10000. In addition, because of the stochastic nature of the MOGA-CB approach we performed 30 runs for each configuration. Table 3.1 summarizes the parameters used in our experiments.

Table 3.1: Experimental parameters.

| Parameter | Value |
|---|---|
| Number of runs per configuration | 30 |
| Number of tasks | 10000 |
| Request submission time | 0.1 to 1 of the scheduling round |
| Request arrival rate $\lambda$ | 15 per scheduling round |
| Request execution time | 2 to 50 scheduling rounds |
| Satisfaction rate | 0.1 to 1 of the best solution value |
| Profit rate | 0.1 to 1 of the best solution cost |
| $\alpha/\beta$ | 9, 3, 2, 1, 1/2, 1/4, 1/8, random |
| Instance types | small, large, extra large |
| Instance performance indexes ($PI$) | 1, 3.98, 7.12 |
| Instance prices | Amazon EC2 pricing history |

#### 3.4.2.1 Pareto MOGA-CB algorithm parameters

The core of the MOGA-CB scheduler is based on a multi-objective genetic algorithm MOGA-CB. Table 3.2 depicts the different parameters of this MOGA-CB algorithm.

Table 3.2: MOGA-CB algorithm parameters.

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

### 3.4.3 Performance evaluation

We have quoted in Section 1.4.2.1 that, to the best of our knowledge, none of the presented works tackles the task-level scheduling problem on a brokering cloud environment using a Pareto approach. Therefore, we have performed a series of experiments with different parameters and configurations to validate the approach that we propose in this chapter. In addition to optimizing the prices and the response times of the tasks using the MOGA-CB, our proposed approach has to return among the proposed Pareto set of solutions the solution that meets at the best the client's satisfaction. To evaluate our contribution we conducted different experiments to study the behavior of our approach according to different parameters. We conducted experiments in average over all the scheduling rounds to study the general algorithm behavior. Moreover, we also did a real time analysis of the results according to the evolution of the criteria. We carried out our experiments on 5 different configurations of tasks. The obtained results were very nearly equivalent. Thus, we discuss only the most relevant one in the following.

The results for each configuration of tasks with its 10000 tasks, for each $\alpha, \beta$ combination and for each profit/satisfaction orientation of the MOGA-CB have been obtained using 30 independent runs. Therefore, the reported results are averaged over these runs. The detailed analysis of our approach is presented in Figure 3.10 to Figure 3.15.

The analysis of our approach will be depicted in four parts. The first part discusses the average result of the objectives at the end of the processing time. The second and third parts show the evolution of the addressed objectives respectively with both disabled and enabled profit parameter. The fourth and last part, deals with the scheduling duration of MOGA-CB. Note that in all the figures that we present below the value representing the satisfaction of the client is in fact his/her disappointment. Therefore, a high disappointment value represents a weak satisfaction and *vice versa*. The objective is to reduce the client's disappointment to increase his/her satisfaction.

- ***MOGA-CB average behavior:*** Figure 3.10 shows that the obtained results over the parameters (profit and disappointment) do not vary when using our Pareto approach. In other words, the algorithm gives roughly the same results over all the ($\alpha$, $\beta$) settings thanks to the Pareto approach. Thus, since the

results are Pareto optimal, the ($\alpha$, $\beta$) parameters cannot alter the final results which are equivalent regarding both satisfaction and profit. The only parameter that varies a little bit is the computation time of MOGA-CB because of the different complexities which may result from a different solution selection in the Pareto set. In addition, Figure 3.11 shows that the cost (price) and the time response objectives complement each other according to the client choice trough the $\alpha$ and $\beta$ parameters. Indeed, when the priority goes to the cost ($\alpha$ higher than $\beta$) the cost is more minimized but leads the time response to suffer from that. The same goes for a high time response priority ($\alpha$ smaller than $\beta$) with opposite behavior. Moreover, we notice that the average results for a random ($\alpha$, $\beta$) configuration gives the same results as for the $\alpha = \beta = 1$ configuration. This is due to the fact that selecting a solution in a Pareto set by promoting randomly each time an objective during several scheduling rounds leads to select a solution equivalent to the one obtained with a policy that does not favor any of the objectives. We can deduce then from the invariability of the satisfaction and the profit results and the similarity of the cost and response time values for the random ($\alpha$, $\beta$) and $\alpha = \beta = 1$ configurations that our Pareto MOGA-CB helps to dispense the client from providing those parameters. Using the Pareto MOGA-CB gives birth to a new satisfaction model leading to the best satisfaction without using the $(\alpha, \beta)$ parameters presented in Equation (3.6).

$$MaxSatisfaction(p,t) = Min\sqrt{p^2 + t^2} \tag{3.6}$$

Besides, in the experiments for an enabled profit broker option, the clients' satisfaction is highly affected and the disappointment increases along with the increase of the margin of the broker. This proves that there is no interest to use this option except in emergency cases, especially because of the good obtained profit results when disabling this option. A deeper analysis of this case where the profit option is enabled is proposed in the following.

- ***Default MOGA-CB real time behavior:*** in Figure 3.12b, we observe the interest of using a Pareto approach to tackle the brokering problem. Indeed, we notice in this figure that MOGA-CB compensates the increase of instances (spots) cost by a reduction in the instances' response time. The two graphics corresponding to both objectives are complementary, thus the increase of the first leads to the decrease of the second and *vice versa*. In addition, as expected the broker's profit is inversely proportional to the instance price. Therefore, to preserve the satisfaction of the clients, the broker's profit decreases when the instances cost increases (less profit margin). Figure 3.13 presents three different values using a logarithmic scale to show the relationship between the number of pending tasks, the average instance spot cost and the average instance response time during the different scheduling rounds. Therefore, the depicted graph shows that there is a tight relation between the number of un-

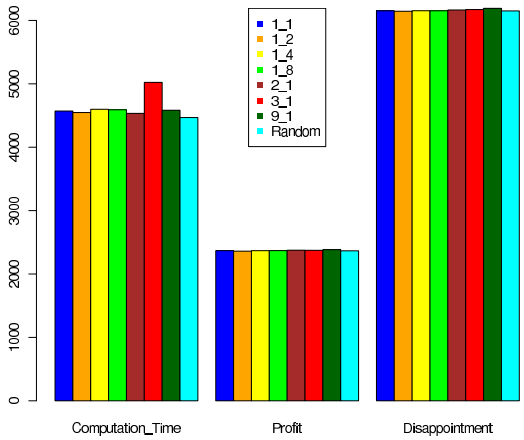



Figure 3.10: The none impact of the $\alpha$,$\beta$ parameters on the results of the Pareto based MOGA-CB algorithm.

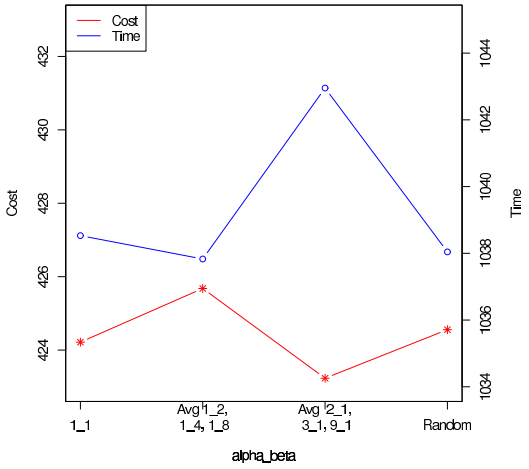
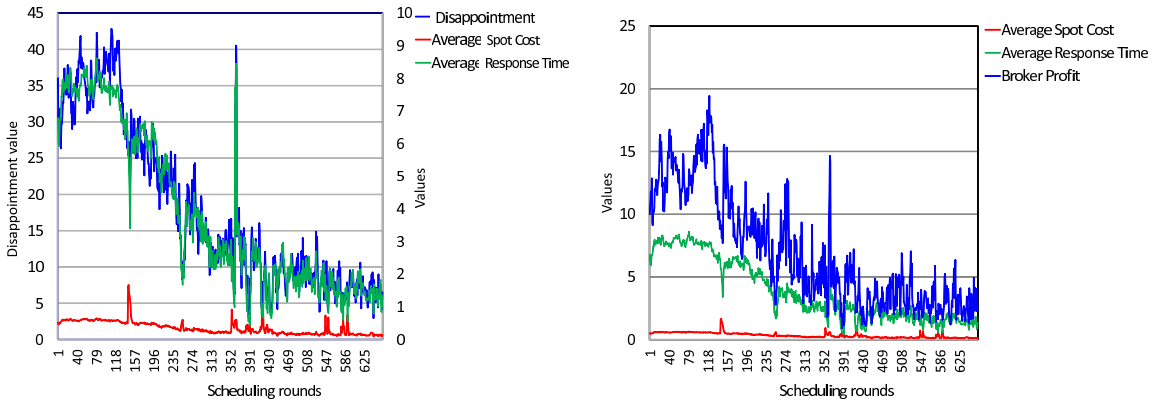



Figure 3.11: The tradeoff relationship between the cost and the response time objectives according to the $\alpha$, $\beta$ parameters.




(a) (b)

Figure 3.12: A disabled broker profit option study over the scheduling rounds of the relationship between the average instances (spots) cost, their average response time and (a): the real time client's disappointment (satisfaction). (b): the broker profit.

finished tasks and the instance response time. Indeed, the less pending tasks they are the faster is the response time of the instances. Conversely, the number of unfinished tasks is inversely proportional to the instances cost. This can be explained by the fact that increasing the price of the instances means in general better instance performances and thus, a better response time and less waiting tasks. In Figure 3.12a, the graphic indicates that the client's disappointment and therefore his/her satisfaction is much more impacted by the response time of the instances than by their prices. This is explained by the fact that the variation of the instances' response time is more significant than that of their cost. Indeed, the cost (price) value of the instances (e.g. see Figure 3.8) are not high, thus, the variation of those costs is less important as well. However, this slight variation in the instance's cost when due to an upgrade in the type of selected instance causes a strong performance improvement which impacts highly the response time of the instance. In addition, unlike the instance's price, the amplitude of the response time criterion is wide. It depends on both the type of instance selected by the algorithm and the current tasks load. One can see then that a response time faster by few seconds is much more important for the client's satisfaction than a price cheaper by few cents.

- ***MOGA-CB profit-oriented real time behavior:*** Figure 3.14b, where the profit option is enabled, shows as expected a direct link between the instance (spot) cost values and the broker's profit. Thus, unlike for the default MOGA-CB settings, the profit increases strongly with the increase in the spot cost. The rate of this increase depends on the percentage of profit required by the broker. Moreover, in this case, we note that the instance's response time cri-

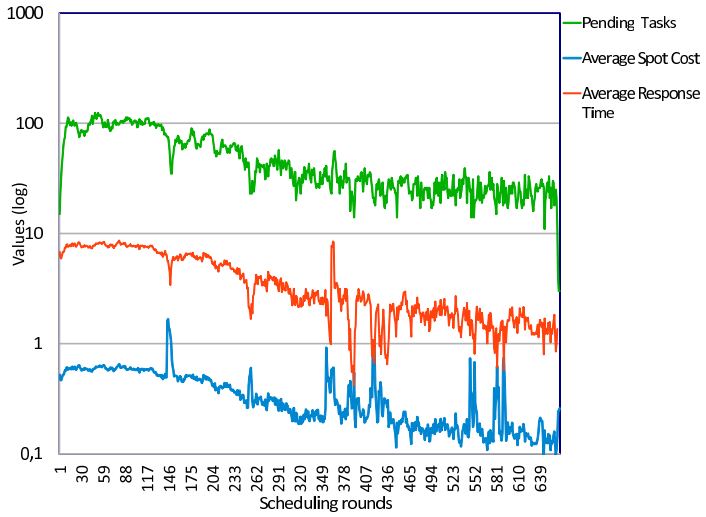
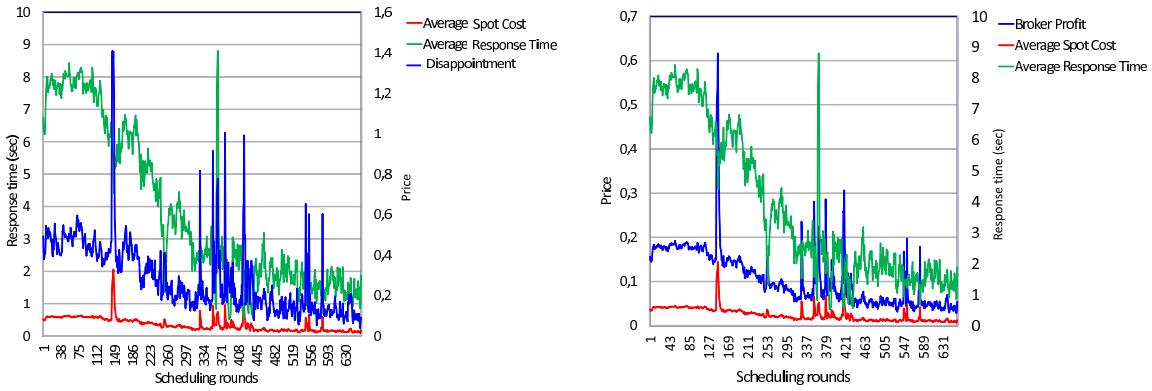



Figure 3.13: The relationship between the number of unfinished tasks, the average instances (spots) response time and the average instances (spots) cost over the scheduling rounds.

terion has no impact on the broker's profit. Moreover, in Figure 3.14a the activation of the profit parameter has negative impact on the client's satisfaction. Indeed, each slight increase of the instance's price, skyrockets the disappointment of the client. This is due to the margin that the broker takes when activating this option in MOGA-CB that affects directly the charged costs of the tasks. Furthermore, we notice that unlike for the satisfaction-oriented settings of MOGA-CB (default settings), the disappointment or the satisfaction of the clients does not depend anymore on the instance's response time but on the price of the instance. This latter becomes the predominating criterion in the client's satisfaction because of the effect that has the profit of the broker over the charged price of the client.

It also should be noted the general decrease in all the curves of the presented graphics during the course of the interval times. This is mainly due to the decrease in the number of unfinished tasks as the algorithm proceeds.

- ***MOGA-CB computation time:*** in this last part illustrated in Figure 3.15, we prove that MOGA-CB never exceeds 30 seconds in its computation time of the solution whatever is the load in terms of requests over all the scheduling rounds. This result is interesting since we know that the schedulers waiting time called *scheduling cycle* or *scheduling round* is about 30 seconds. Hence, the computation time of our algorithm never exceeds this threshold, in addition to be covered by the latter. We note also different variations in MOGA-CB processing time such as around the 149 scheduling round. This is due to the irregularity in the arrivals of tasks since following a Poisson distribution.




(a) (b)

Figure 3.14: An enabled broker profit option study over the scheduling rounds of the relationship between the average instances (spots) cost, their average response time and (a): the real time client's disappointment (dissatisfaction). (b): the broker profit.

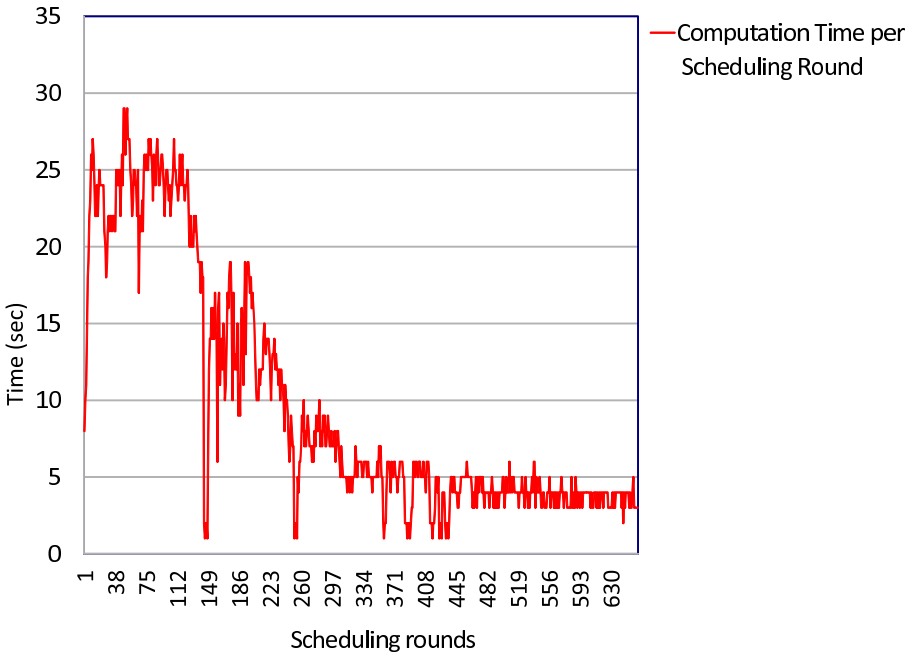



Figure 3.15: The computation time duration of the MOGA-CB algorithm over each scheduling round.

## 3.5 Conclusion

In this chapter, we presented a new approach for the second level of scheduling in cloud computing, the task-level scheduling. This contribution has been applied to a cloud brokering environment using a multi-objective genetic algorithm MOGA-CB. The main challenge at this level of scheduling is the minimization of both the response time and cost (price) of the tasks at the same time. In that purpose, optimizing those two criteria using a Pareto approach, leads to address both the client's satisfaction and the broker profit. This was made possible by exploiting the instances' (spots) cost fluctuation and their performances' heterogeneity. The main contributions of this chapter are the following:

- *Pareto MOGA-CB:* we proposed for a cloud broker a multi-objective genetic algorithm for task-level scheduling. The idea is, unlike other approaches in the literature, to tackle the common variation criteria of both the client' satisfaction and the broker's profit instead of dealing with them directly. Thus, we dealt simultaneously with the response time and the cost of the VM instances criteria to address the satisfaction and profit issues. We also designed MOGA-CB to fit the scheduling constraints such as processing time by respecting the scheduling round duration.

- *Criteria relationship analysis:* we performed a deep analysis on the different behaviors of MOGA-CB and the obtained results through different configurations. We highlighted the relationship between the satisfaction/profit results and the impact on them of the cost and response time of the VM instances. We proved that enabling the broker profit parameter in our algorithm decreases significantly the client's satisfaction. We also showed that the satisfaction is more related to the instances' response time than to the instances' cost using MOGA-CB with default parameter settings.

- *New selection model:* we improved through the Pareto multi-objective genetic approach the usage of the utility theory in economics [Mankiw 2008] for the client's satisfaction modeling. Indeed, the results showed that our Pareto-based approach helps to dispose from the client's requested $\alpha$ and $\beta$ parameters. To do so, it provides a Pareto optimal or near optimal solution allowing a tradeoff between the response time of the VM instances and their cost by selecting from the Pareto set the nearest solution to the origin of the axis.

Chapter 4

# VM-level Scheduling on top of a Cloud Manager

### Main publications related to this chapter

- Y. Kessaci, N. Melab, E-G Talbi, A Multi-start Local Search Heuristic for an Energy Efficient VMs Assignment on top of the OpenNebula Cloud Manager, *Future Generation Computer Systems Journal*, 2013.

- Y. Kessaci, N. Melab, E-G. Talbi. An Energy-aware Multi-start Local Search Heuristic for Scheduling VMs on the OpenNebula Cloud Distribution. *International Conference on High Performance Computing and Simulation (HPCS)*, Spain, Madrid, 2012.

### Contents

## 4.1 Introduction

The cloud computing paradigm has been designed around the client's service, it is therefore natively market-oriented. Thus, as presented in Section 1.4.1.2, two main optimization levels related to the market-oriented cloud have been raised and addressed in Chapter 2 and Chapter 3. However, the optimization related to both the service-level and the task-level optimization is not sufficient and is tightly related to a lower-level optimization. Indeed, the obtained results of both the market-oriented techniques face the issues related to the hardware configuration and the management policy of the used infrastructure.

Cloud managers are the tools in charge of these issues mainly using scheduling as an optimization tool. However, the scheduling approaches proposed in many cloud managers are limited regarding the criteria taken into account. Indeed, despite the increasing impact of the energy on many applications [Koomey 2008], cloud managers are still rarely designed with the objective to optimize the energy consumption reduction. Moreover, dealing with energy criterion is not easy. Thus, reducing energy is generally made at the expense of performance which goes against the principle of cloud computing. Therefore, addressing these tricky issues is mandatory in order to satisfy the upper scheduling levels and at the end the client. Thus, a low level VM-level scheduling is necessary to optimize the assignment of the Virtual Machines (VM) to the physical resources of the infrastructure.

In this purpose, we present new approaches that tackle both the energy consumption and the VMs performance issues within a realistic cloud infrastructure. The first scheduler that we propose (EMLS-ONC) is based on a multi-start local search metaheuristic that provides a near-optimal or optimal scheduling by mapping the incoming VMs according to the minimum energy consumption. We also present a Pareto bi-objective version of this scheduler (EMLS-ONC-MO) that addresses simultaneously both the energy consumption criterion and the performance of the VMs in terms of response time to maintain the performance level of the infrastructure. Note that both EMLS-ONC and EMLS-ONC-MO aim to achieve the previous cited objectives always by assigning the maximum number of VMs. To be as thorough as possible and to fit the reality we embed both our schedulers within a real cloud manager. We choose as a case study the popular OpenNebula cloud manager as our software management solution. In addition, the experiments that we present to validate our algorithms are done on both artificial and real cloud infrastructures with different VM workloads.

The remainder of the chapter is organized as follows. In Section 4.2 we present the application, system, energy and performance models investigated in our problem modeling. Our approaches are presented in Section 4.3. The results of our experimental study are discussed in Section 4.4. The conclusion is drawn in Section 4.5.

## 4.2 VM-level Scheduling Model

### 4.2.1 Cloud managed model

Following the previous IAAS model presented in Section 2.2.1 and Section 3.2.1, the model in this chapter is based on the same model but focuses on its lowest level. Indeed, we tackle the scheduling problem at the level of the provided VM instances (spots) cited in Chapter 3. The aim is to assign those VMs to the physical machines of an infrastructure managed by the OpenNebula [1] cloud manager. The features of OpenNebula are presented in Table A.1. The cloud considered in our model can vary from a set of few installed hosts to a multi-cluster distributed cloud.

We model this problem as a two-tier architecture composed of a cloud infrastructure provider and users. The users have access to the cloud resources by requesting them from the provider. The service proposed by the cloud provider in our approach offers those resources as VMs to the users. The role of our approach is to help the provider to optimize the scheduling of those VMs on the physical resources by addressing two criteria (energy, VM performance) in his/her cloud management policy. We use for this a multi-start local search metaheuristic. The location of the default OpenNebula's scheduler and its replacement by our EMLS-ONC (or EMLS-ONC-MO) scheduler is shown in Figure 4.1.

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

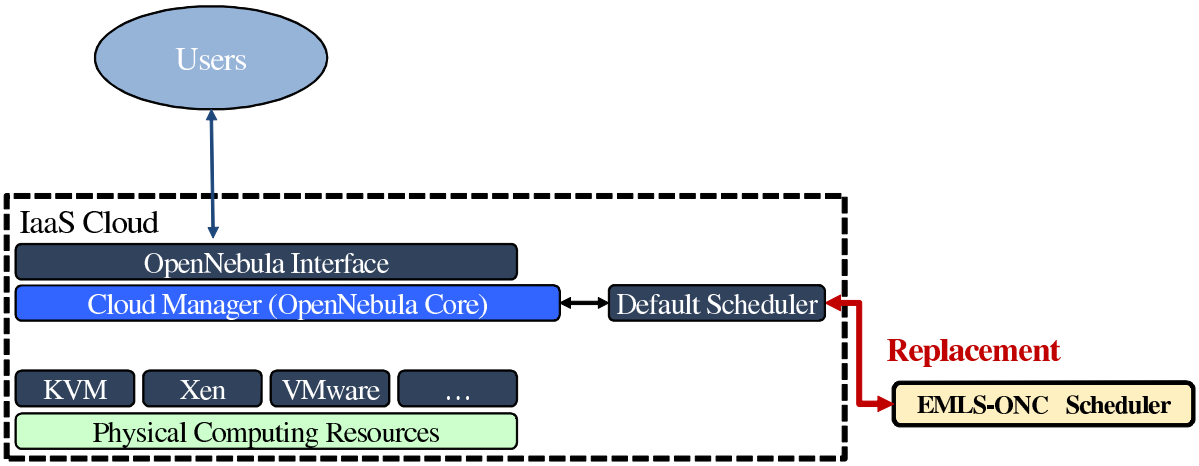



Figure 4.1: Overall architecture of OpenNebula including the EMLS-ONC (or EMLS-ONC-MO) scheduler location.

[1] http://www.opennebula.org/

### 4.2.2 Energy consumption model

The energy model of a federation cloud such as presented in Section 2.2.2 is directly deduced from the consumption of the hosts that compose each cloud of the federation. Therefore, the energy model of a single cloud results also from both IT and auxiliary equipments. Our approach is computation-oriented, hence the most of the energy consumption is due to the computation. Therefore, this energy is the one considered in our model. That leads to consider also the impact of the cooling energy consumption. Indeed, cooling energy is significant and directly related to the energy consumption of computation. Moreover, we 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 the entire processor infrastructure with and without the DVFS feature. Our scheduler aims to prove the contribution of the hardware heterogeneity (frequency, CPU usage, etc) offered by the cloud to the energy reduction.

The processor energy model is derived as for the contribution of Chapter 2 from the power consumption model in Complementary Metal-Oxide Semiconductor (CMOS) [Burd 1995, Pillai 2001]. After expressing the voltage value in a linear form with the CPU frequency such as done in [Chen 2005b, Wang 2008], we obtain the Equation (4.1).

$$P = \alpha f^3 + \beta \tag{4.1}$$

Regarding the cooling energy consumption, it is based on the following proved assumption. This assumption is based on the observation showing that the higher the CPU usage is (see Figure 4.2), the higher the CPU's temperature is (see Figure 4.3) and faster the cooling fan turns (see Figure 4.4). In all the figures, the blue and the red lines designate the information related to the devices usage and temperature respectively with and without using the cool'n'quiet technology [AMD 2004]. As a result of the observation from the figures, we deduce that the CPU usage is a major parameter mainly regarding its impact on the cooling devices. It also learns a lot about the system energy behavior. Since our model can handle up to 100 VMs in each scheduling cycle (see Section 4.3.3), this can quickly become CPU intensive for the hosts that compose 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 chapter, we only consider the CPU related 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 contribution is not to raise the energy saving improvement. Instead of that, the aim is to propose a thorough energy model by combining both the frequency and the CPU usage of the hosts in the objective function of the EMLS-ONC algorithm to benefit from the heterogeneity offered by the hosts. The scheduler retrieves the processor frequency and the current CPU usage of each host by requesting the hypervisor. The used

hypervisor in our model is KVM and the objective function of the energy criterion is defined in Equation (4.2).

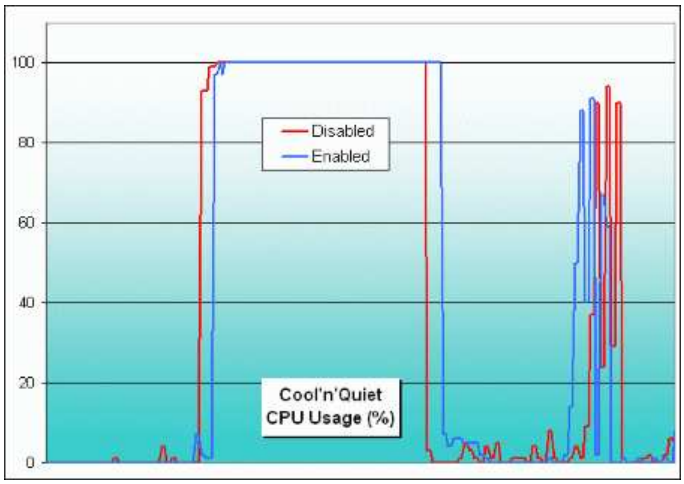



Figure 4.2: CPU usage during a stress period (x-axis).

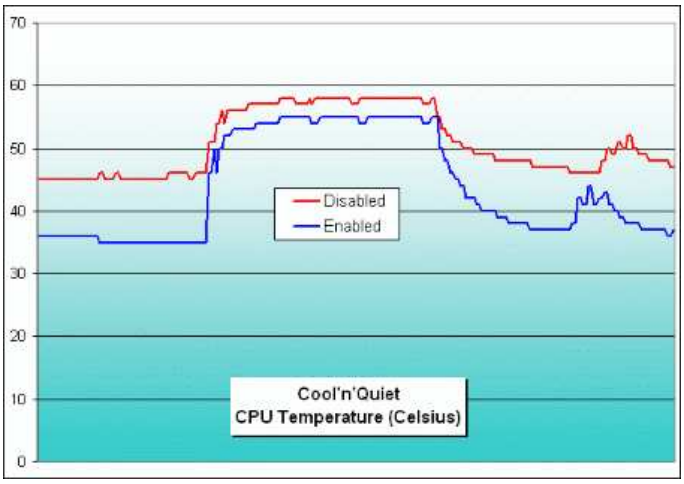



Figure 4.3: CPU temperature during a stress period (x-axis).

$$(E)_{ij} = (\alpha_i f_i^3 + \beta_i) \times e_j \times n_j \times (\frac{CPU\_usage_i}{const\_value} + 1) \tag{4.2}$$

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. Indeed, according to the current CPU usage of the host the consumed energy may vary. The $const\_value$ represents the level of this variation, it is a constant value that gives the coefficient of the energy increase according to the CPU usage increase. In other words, $const\_value$ represents the ratio between the

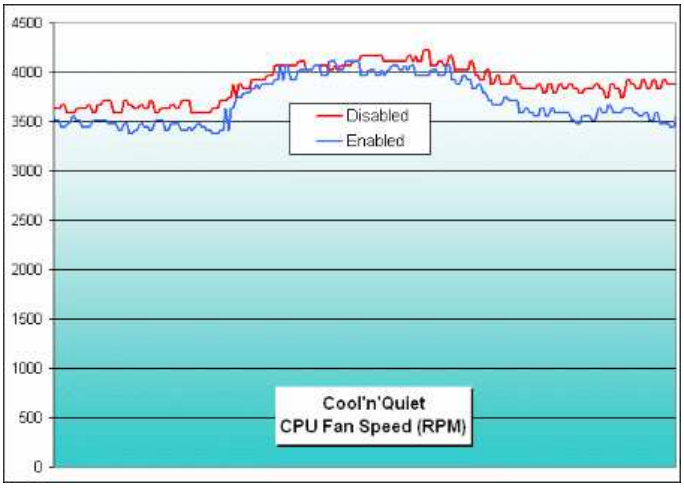



Figure 4.4: FAN rpm during a stress period (x-axis).

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 4.5) to calibrate 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 of both its common use in data centers and its design for CPU intensive usage. We stressed the processor 5 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 thus decided that in our objective function the *const_value* equals 3, which represents 33% of extra power between the idle and the maximum processor usage.

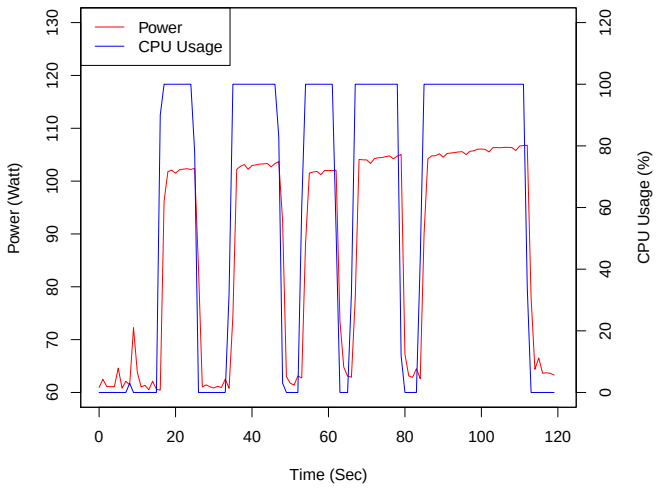



Figure 4.5: CPU usage vs energy consumption.

### 4.2.3 VM performance model

Reducing the energy consumption in a cloud infrastructure is a challenging issue. However, if naively addressed this could conduct to some drawbacks in terms of performance. Indeed, the virtualization tool offered by the cloud allows different VMs to share the same physical host. A trivial way to deal with energy consumption reduction is 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 issue is tricker 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, the VMs are not isolated anymore [Verma 2008]. In [Verma 2008], the experiments on the VM isolation show that for the VMs with high memory needs, the response time (delays) of the VMs increases due to the cache misses. We noticed that this VM response time doubles between none back ground utilization with 0% value (lonely VM) and a fully back ground utilization with 100% value (full physical host). We notice also that the increase of the response time is linear following the memory increase. From this behavior, we deduce Equation (4.3) to link the VM response time (VM performance) $Response\_time_{ij}$ to the VM memory needs and the memory usage of the host.

$$Response\_time_{ij} = Memory_j + Memory_j \times Mem\_usage_i \tag{4.3}$$

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 $i$.

### 4.2.4 Cloud manager based VM-level scheduling problem modeling

As illustrated in Figure 4.6, the focus in this chapter is made on a two-tier of our VM-level scheduling model. The first tier is a cloud provider which has $N$ heterogeneous hosts (data centers). The second tier is a set of users with $J$ VM requests for running their applications. The problem consists of scheduling $J$ VMs on $N$ data centers. As for all the different scheduling levels presented in the previous chapters, since the scheduling problem is NP-hard [Garey 1979] the VM-level scheduling problem is NP-hard as well. Thus, a metaheuristic algorithm appears to be the most appropriate approach to adopt to deal with the problem. The metaheuristic that we used is a multi-start local search. This metaheuristic is composed of two parts. First, the multi-start part that brings diversification in the problem offering a bigger exploration. The second part consists of each local search that adds accuracy in the solution processing using the intensification. Hence, our approach provides both diversification and intensification of an evolutionary

approach while fitting the cloud manager constraints. In addition, EMLS-ONC (or EMLS-ONC-MO) always returns the assignment within the time limit of the *scheduling cycle* (see Section 4.3.3).

With OpenNebula, the user submits VM requests with requirements. Those VM requirements are the number of CPU, memory size, storage capacity, the type of the operating system, etc. In our problem, we added a time requirement in the definition of the VM to know the duration of the VM execution in order 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 of the 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 $(m_j)$. All those information except the reservation time parameter, are mandatory to build even a most basic VM. Regarding the reservation time parameter, as aforementioned, it is useful in our model 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 running on it.

The objective function to be minimized in our approach deals with, 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. The two objectives are formulated in Equation (4.4) and Equation (4.5):

$$Minimizing\ the\ energy\ consumption = \sum_{i}^{N}\sum_{j}^{J}(E)_{ij} \tag{4.4}$$

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 $i$.

$$Maximizing\ the\ VMs\ performance = Minimizing(\sum_{i}^{N}\sum_{j}^{J} Response\_time_{ij}) \tag{4.5}$$

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 following a Pareto approach, 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.

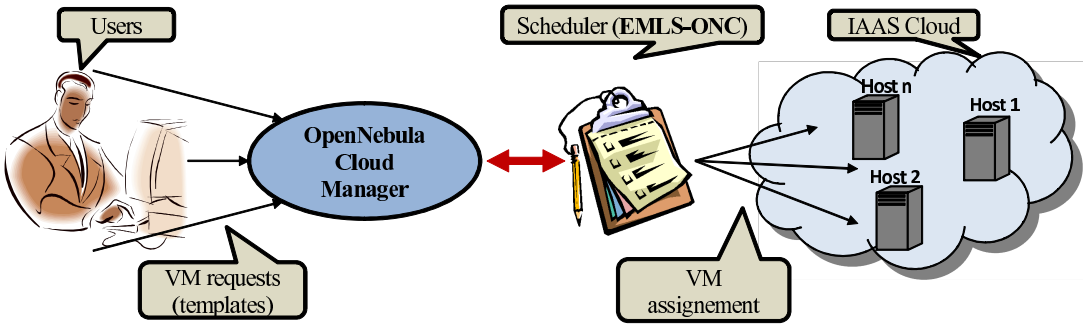



Figure 4.6: Two-tiers cloud model representing the user tier, the provider tier (Cloud), and the relationship between both OpenNebula and EMLS-ONC (or EMLS-ONC-MO) scheduler.

## 4.3 The Proposed Multi-start Local Search Algorithm for VM-level Scheduling

Before describing our EMLS-ONC and EMLS-ONC-MO approaches, we describe briefly in the following sections both algorithms that we compared with our approach: the default OpenNebula scheduler and two bin packing FFD-based algorithms.

### 4.3.1 OpenNebula scheduler heuristic

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

### 4.3.2 Bin packing FFD-based scheduling heuristics

The FFD-based scheduler as its name suggests is based on the First Fit Decease heuristic (FFD) such as the work [Verma 2008]. As said in Section 1.4.2.2, the algorithms that are based on FFD bin packing have been proven to perform the best for energy reduction purposes [Mills 2011]. The idea is to sort the pool of VMs to be assigned in a decreasing order (i.e. from the one with the most requirements to the one with the least) and to assign those VMs each time to the server with the lowest energy consumption according to Equation (4.1). 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 been implemented. This version assigns the sorted VMs to the servers with the most important memory capacities. This helps to avoid a memory overload which will lead to cache misses and therefore to a decrease in the VMs performance. The Pareto-based EMLS-ONC-MO has been compared to both energy- and memory-aware bin packing FFD algorithms.

### 4.3.3 Single objective EMLS-ONC and Pareto-based EMLS-ONC-MO steps

Both 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 addressed objectives. Before each scheduling, EMLS-ONC or EMLS-ONC-MO waits for a fixed period of time called *scheduling cycle*. This period allows one to gather a pool of VMs in order to get 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 (LS) 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 based on 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 LSs are sufficient to get a good solution. However, the drawback of relating the number of launched LSs to the number of hosts lies in the processing time. A tradeoff has been found through experiments to bound their number to 20. After the end of all LSs, all the best solutions they provide are compared. Only the best of them is kept and used, at the last step by OpenNebula to assign the VMs. Thereafter, the states of the hosts are updated and a new scheduling cycle is started. Figure 4.7 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 7). After the end of all the processes of all the LSs, each LS updates a common (global) archive with the elements of its private archive to get the final global archive of non-dominated solutions using the same Algorithm7. The penultimate step of the flowchart (Best scheduling step) 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, this same step represents the best energy efficient solution among the solutions of each LS.

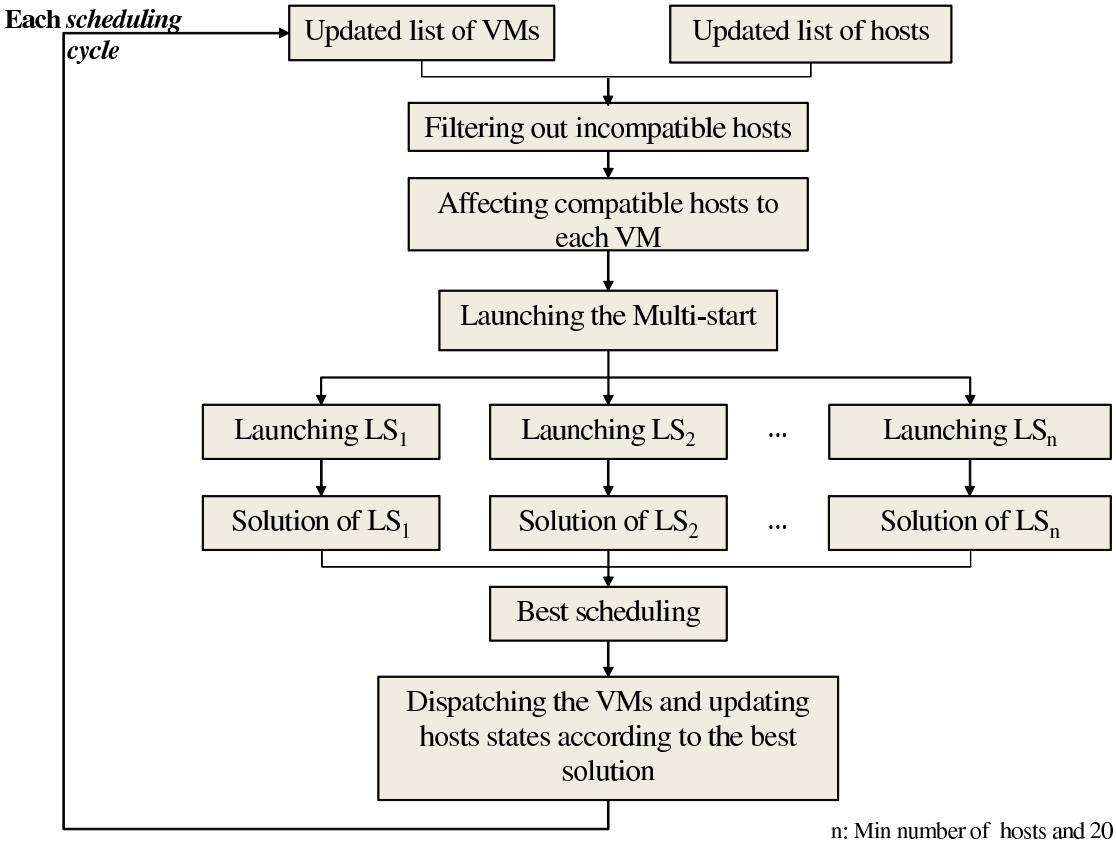



Figure 4.7: The Flowchart of the EMLS-ONC scheduler.

### 4.3.4 VM-level scheduling 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 in Figure 4.8.

Figure 4.8 represents one possible assignment among plenty proposed by the multi-start local search algorithm. In the proposed example we identify two major information: the first row of the table (map keys) contains the VMs that are assigned

**Algorithm 7** Local Search Pareto archive ranking.

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

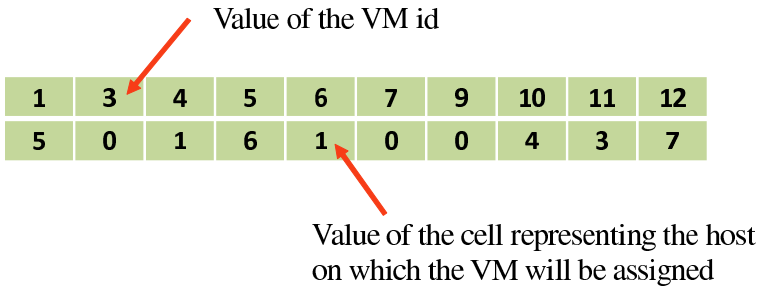



Figure 4.8: VM-level problem encoding.

and the second row identifies the hosts to which the VMs are allocated. In other words, the first column represents the first VM of the pool that is currently being treated by EMLS-ONC (or EMLS-ONC-MO). The VM is identified with the ID 1 and 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 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. This encoding may be wrongly considered as similar to the one presented in both Section 2.3.2 and Section 3.3.2 for respectively the service-level scheduling and the task-level scheduling. Therefore, the reason for choosing a map instead of a simple vector is to fit the encoding of the pretreatment of the OpenNebula cloud manager. Indeed, as shown in Section 4.3.3, the latter filters the hosts and assigns to each VM a list of potential hosts in a vector. Moreover, the VM's IDs are not in a sequential order. Therefore, to be able to retrieve both the information and the list of hosts of each VM we need to store the IDs as map keys.

### 4.3.5 Solution initialization

The initial solution in a local search algorithm may affect the whole exploration of the landscape. In fact, this step is quite important and affects the quality of the future results. In our proposed approach, since we deal with a multi-start method,

each local search execution has its own initialization method and then its proper initial solution. The initialization 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 hosts. 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.

### 4.3.6 EMLS-ONC/EMLS-ONC-MO algorithm

In Section 1.3.2.2, we have presented the general concepts of the single-based solution metaheuristics. The local search algorithm is one of the best known S-metaheuristic. Its role is to generate a number of candidate solutions from the initial solution using neighborhood operators in order to find the best assignment according to the specified objective(s). The multi-start parallel method allows launching several LSs to add diversity and robustness, since each LS provides intensification. LS starts by generating the initial solution according to the process explained in Section 4.3.5. 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 mechanism. The first operator is dedicated to generate neighborhoods for small cloud configuration or small number of VMs, while the second is dedicated to large neighborhoods with large clouds and numbers of VMs. A cloud is considered as small when it contains less than 50 hosts while a number of VMs is small when it arrives less than 5 VMs per scheduling cycle.

The first operator (see Figure 4.9) 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 less energy for the EMLS-ONC or has the best Pareto solution for EMLS-ONC-MO. In the second operator, the number of both hosts and VMs is bigger. Therefore, the algorithm cannot afford to enumerate all the potential solutions 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 (i.e. one enumeration of one VM hosts set) generates one neighborhood. LS ends when all the hosts of the hosts set of all the VMs are enumerated with the first operator, or all the hosts of a selected range in the hosts set of each VM are enumerated with the second operator. Each solution of the

**Algorithm 8** Single-objective Local Search algorithm.

```
1: Input: A set of Hosts and VMs
2: InitSolution(Hosts, VMs, initialSolution);
3: for all VMs of currentSolution do
4:    if #Hosts < 50 or #VMs < 5 then
5:       apply neighborhood operator 1;
6:    else
7:       apply neighborhood operator 2;
8:    if currentSolution is feasible and currentSolution < bestSolution then
9:       currentSolution:= bestSolution
10: Output: The best solution (scheduling)
```

**Algorithm 9** Pareto Local Search algorithm.

```
1: Input: A set of Hosts and VMs
2: InitSolution(Hosts, VMs, initialSolution);
3: currentSolution:= initialSolution
4: for all VMs of currentSolution do
5:    if #Hosts < 50 or #VMs < 5 then
6:       apply neighborhood operator 1;
7:    else
8:       apply neighborhood operator 2;
9:    if currentSolution is feasible then
10:      call Pareto archive ranking;
11:   currentSolution:= select randomly solution in Pareto archive
12: Output: A set of Pareto non-dominated solutions (scheduling)
```

generated neighborhood is checked for its feasibility. A fitness value is also assigned to this solution. Moreover, in order to speed up the computation of the proposed EMLS-ONC and EMLS-ONC-MO algorithms and to respect the scheduling cycle time constraint of the OpenNebula cloud manager, each new evaluation of the fitness of a modified solution relies on a delta evaluation. A delta evaluation helps to evaluate only the assignment of the concerned VM in the new generated solution. That way, it avoids an extra time due to the evaluation of the already known fitness values of the other unmodified VMs.

The best found solution from 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 LSs in the multi-start approach, the choice of this parameter is due to the complexity of the problem. Indeed, the more VMs they are the more iterations are needed to find a good solution. The pseudo-code of each LS of both EMLS-ONC and EMLS-ONC-MO is respectively presented in Algorithm 8 and Algorithm 9.

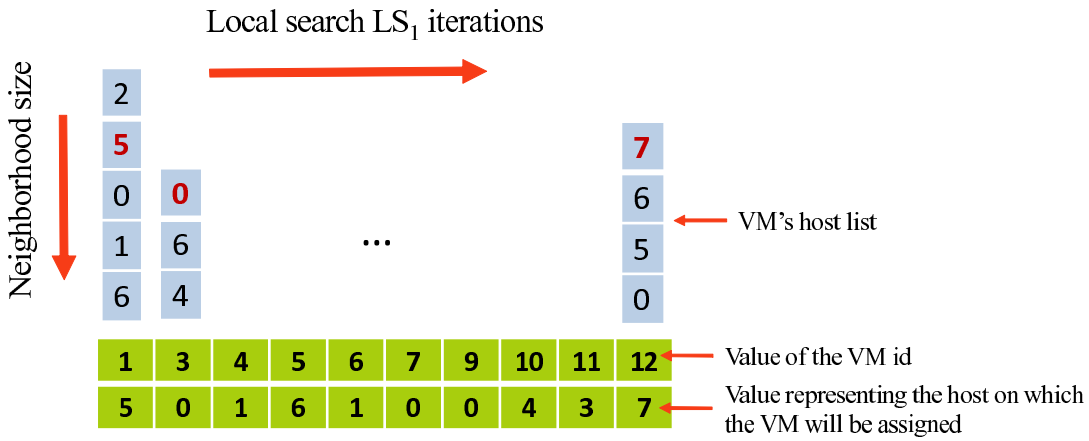



Figure 4.9: The Mechanism for generating the neighborhood from an initial solution (neighborhood operator) of one of the local search algorithm that composes the EMLS-ONC/EMLS-ONC-MO scheduler.

## 4.4 Experimental Study

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

### 4.4.1 Experimental settings

The experiments have been conducted using two types of settings. The artificial sittings using different generated VMs and hosts and the real settings using real VMs and cloud infrastructures.

#### 4.4.1.1 Artificial hardware settings

- **VMs' settings:** regarding the inputs of the scheduler, we have generated VMs in an XML format. Thus, the OpenNebula parser reads their associated parameters in a realistic way. These latter vary in our experiments according to three parameters. Indeed, as said in Section 4.2.4 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 have generated randomly this triplet where the execution time $e$ is an integer value from {1, 2, ..., 10} hours, the processor requirement $n$ varies in the set $\{k/2; 1 \leq k \leq 18\}$ CPUs and the memory needs $m$ is an integer value in {1, 2, ..., 8} GBs for the CPU-intensive VMs and {1, 2, ..., 15} GBs for the memory-intensive VMs.

- **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 and 24 cores, its memory capacity from [2, 24] GB and its CPU frequency from [1, 3] Ghz. During the generation of the different features of a host, we ensure the validity of the information about the amount of CPU and memory used resources of each host, 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 ranges of VM parameters presented above are deduced from the type of VMs proposed by the cloud providers like EC2 for Amazon [Ama 2012b]. Indeed, as previously said, the VM instances that are scheduled in this chapter are the ones proposed to the clients to run their tasks in Chapter 3. The VM instances proposed in EC2 [Ama 2013] vary from small ones (1 CPU, 1.7 GBs memory) to extra large ones (8 CPUs, 15 GBs memory). Note that the existence of floating values as 0.5 for the parameter $n$ of the VM's CPU needs are due to virtualization that allows affecting less than a whole physical core to a VM (e.g. tiny VMs). The execution time values range between 1 and 10 hour(s). Those values are the common ones in terms of reservation. They oscillate between a short and a long reservation.

Regarding the interval values of the hosts they range between a personal 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*.

#### 4.4.1.2 Real hardware settings

For the real experiments we generated real VM templates for deployment, as a real user has to do when he/she fills the VM template. The VMs parameters (execution time, number of CPUs and memory) were generated following a Poisson distribution with respectively the $\lambda$ parameters (1, 2, 4). Each VM distribution sample is composed of 12000 values. The used values 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 [GRI 2013]. We used the site of Nancy equipped with clusters allowing the virtualization to deploy the OpenNebula cloud manager with and without our proposed algorithms. The hardware configurations of the hosts that compose our cloud are summarized in Table 4.1.

Table 4.1: Hardware specifications of the hosts.

| Device Configuration | 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 |

### 4.4.2 Parameter settings for EMLS-ONC/EMLS-ONC-MO and instance types

The EMLS-ONC and EMLS-ONC-MO schedulers are proposed to be integrated in a cloud manager such as OpenNebula. Therefore, they have to be flexible and fit different cloud configurations. In this context, we conducted our experiments on both algorithms during a fixed number of *scheduling cycles* for each instance. We define an instance as a fixed number of VMs that arrive per scheduling cycle on a given number of hosts. In these experiments, we compare the EMLS-ONC with an energy-aware algorithm (Energy_FFD), while the multi-objective EMLS-ONC-MO is compared to both energy-aware (Energy_FFD) and VM performance-aware (Memory_FFD) algorithms. Note that comparison of both proposed algorithms is also done with the default OpenNebula scheduler (OpenNebula). We measure in these experiments the ability of the algorithms to handle a flow of VM requests arriving at the same time (one scheduling cycle) by a cross comparison of the obtained results for 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 only with the scheduling process part of the algorithm, they do not give interest to the physical VMs dispatching phase. The VM dispatching phase is handled by OpenNebula.

#### 4.4.2.1 Artificial experimentation parameters

In our artificial experiments, we used some parameters: the scheduling cycle duration, the number of scheduling cycles in row, the 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, ranging from a small local cloud with 5 hosts to a wide distributed cloud with 320 machines. For the variation of

the VMs arrival rates we use 5 different loads from a single VM to a massive arrival of 100 VMs at each scheduling cycle. Each instance is defined to be the couple (# VMs, # hosts) during 20 cycles of 30 seconds. This represents for the highest workload (100 VMs/scheduling cycle) which equals 2000 VMs for the biggest instance of each cloud configuration. The time duration of the *scheduling cycle* 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. Moreover, the number of scheduling cycles in row has been determined empirically. 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). The experiments parameters are summarized in Table 4.2.

Table 4.2: 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 |

#### 4.4.2.2 Real experimentation 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 [GRI 2013]. The VMs arrival workloads during each scheduling cycle are 5, 50 and 100 VMs. The objective of this experiment is to prove the feasibility of the deployment of our algorithm over physical machines, while handling their constraints (lags, communications, ...). The behavior of EMLS-ONC and its performance are compared to default OpenNebula scheduler. We considered all configurations to observe the different algorithm behaviors. We expressed the couples (# VMs, # hosts) as ratios. A ratio is the relationship between the number of VMs to be assigned and the number of hosts that compose the cloud. It is obtained by dividing the number of VMs by the number of hosts. Thus, we dealt with a small ratio (5 VMs, 50 hosts), a big ratio (100 VMs, 50 hosts) and a medium ratio (100 VMs, 200 hosts). The scheduling cycle and the number of scheduling cycles in row have been fixed for the same reason as mentioned in Section 4.4.2.1. We used 30 seconds 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. Besides, the number of scheduling cycles in row was fixed to 10 for the same reason as mentioned in the artificial experimental parameter. Indeed, a value of 10 scheduling cycles in row was sufficient to saturate the 200 hosts cloud. The experiments parameters are summarized in Table 4.3.

Table 4.3: 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 |

### 4.4.3 Performance evaluation

In the following, we discuss the experimental study of the two proposed scheduling metaheuristics, integrating both the energy consumption and the VM performances criteria on the top of OpenNebula cloud manager. We have performed a set of experiments with different parameters cited before in both Section 4.4.2.1 and Section 4.4.2.2 respectively for the artificial and the real experiments.

The first single objective approach EMLS-ONC is compared to two algorithms. The first comparison is done with an energy-aware FFD based algorithm. The FFD algorithm, as said previously, is proved to perform very well in energy savings. The second comparison is 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 performance of the EMLS-ONC over the default OpenNebula scheduler when deployed on real machines.

The second approach, the Pareto EMLS-ONC-MO, is compared to three algorithms. Indeed, dealing with two objectives, we compare EMLS-ONC-MO to two heuristics tackling each one an objective among the two. In addition, we compare 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 4.3.2, 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 stochastic nature of both EMLS-ONC and EMLS-ONC-MO, we run each experiment for each instance 30 times. The reported 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 *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, the *RPC* acronym designates the *Relative Percentage Change*. It represents the difference between the obtained values of our algorithm and the values of the algorithm compared with, expressed in percentage. In our case, since dealing with minimization, negative values mean the improvements of our algorithm while positive values mean worst results. Conversely, since dealing

with maximization regarding the *# of Additionnal Assigned VMS* row, negative values mean worst results while positive ones express the improvement.

In the reported 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 algorithms. Indeed, it is not sufficient to compare the obtained values for each criterion at each scheduling cycle. First, we used the cumulative sum to link the results of all scheduling cycles in order to get the evolution of the addressed criterion through the different VMs arrival waves. Second, the normalization uses the cumulative sum of the criterion and the cumulative number of assigned VMs to give at each moment the value of the addressed criterion regarding the number of assigned VMs.

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 dealing with the energy issue. The second one deals with the Pareto bi-objective EMLS-ONC-MO approach compared with the energy-aware algorithm. The third section focuses on the same Pareto bi-objective EMLS-ONC-MO approach but compared with the VMs performance-aware algorithm. Finally, the fourth section presents the feedbacks obtained from the experiments on a real infrastructure.

Figure 4.10 to Figure 4.19b show the results of 3 most specific instances in order to cover all scenarios. The first for small cloud and small number of VMs (20 hosts, 5 VMs), the second for a medium cloud with a big number of VMs (80 hosts, 100VMs) and the last for big cloud with big number of VMs (320 hosts, 100 VMs).

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

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

- *Number of Scheduled VMs:* Table 4.4 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 process applied by OpenNebula which keeps more space during the successive scheduling cycles for the next VMs arrivals. Indeed, 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 higher 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:* here, we compare the computation times of the approaches. Table 4.5 shows that the processing time of

both the energy-aware FFD algorithm and the default OpenNebula scheduler are significantly smaller than the scheduling cycle time. This is due to the low complexity 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 is really fast for small instances (Min value).

- *Energy consumption savings:* in Figure 4.10 and Figure 4.11 we notice that the EMLS-ONC energy histogram is always lower than the ones of both OpenNebula and the energy-aware FFD algorithm, which means better energy reduction during the whole scheduling cycles. Nevertheless, in Figure 4.13 we notice that after a good starting, 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, EMLS-ONC assigns more VMs in that period of time (see Figure 4.12) which leads to an energy consumption increase. However, the additional assigned VMs through the next scheduling cycles is done efficiently which brings down again 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 observe the real energy efficiency of each approach.

  In addition, one can observe 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 (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. One can also note that the Energy-aware FFD approach 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 4.11.

  Table 4.4 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.

Table 4.4: 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 4.5: 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** |

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

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

- *Number of Scheduled VMs:* Table 4.6 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 worse than OpenNebula for some individual instances but still better in average. This is due again to the consolidation mechanism of OpenNebula 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

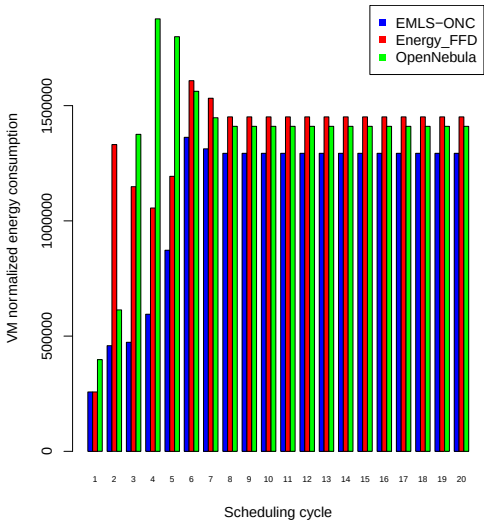



Figure 4.10: Comparison of the results of the normalized energy consumption during 20 scheduling cycles in row between EMLS-ONC scheduler, energy-aware FFD scheduler and OpenNebula scheduler for a configuration of 5 VMs and 20 hosts.

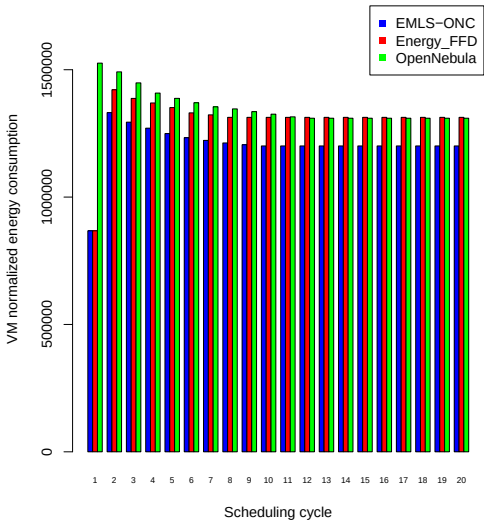



Figure 4.11: Comparison of the results of the normalized energy consumption during 20 scheduling cycles in row between EMLS-ONC scheduler, energy-aware FFD scheduler and OpenNebula scheduler for a configuration of 100 VMs and 80 hosts.

other algorithms mainly on instances with a large infrastructure compared to the number of VMs (# hosts >> # VMs) like for instances (20 hosts, 5

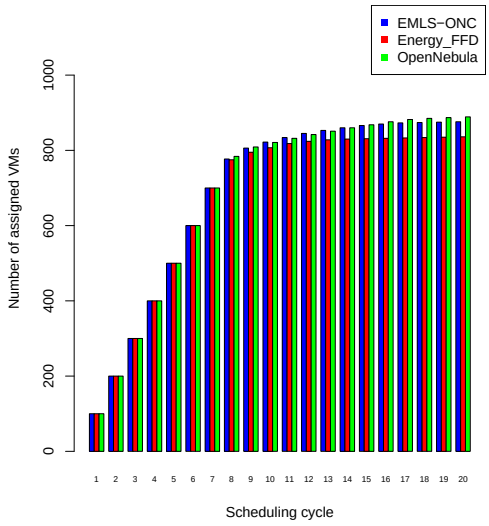



Figure 4.12: Comparison of the number of scheduled VMs during 20 scheduling cycles in row between EMLS-ONC scheduler, energy-aware FFD scheduler and OpenNebula scheduler for a configuration of 100 VMs and 320 hosts.

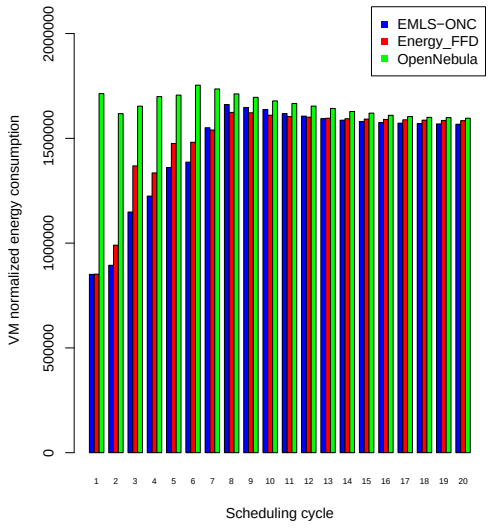



Figure 4.13: Comparison of the results of the normalized energy consumption during 20 scheduling cycles in row between EMLS-ONC scheduler, energy-aware FFD scheduler and OpenNebula scheduler for a configuration of 100 VMs and 320 hosts.

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 4.7 shows that the processing time of both the energy-aware FFD algorithm and the default OpenNebula scheduler are significantly smaller than the scheduling cycle time. This is due to the low complexity 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 4.14a and Figure 4.16a for the energy criterion and, Figure 4.14b and Figure 4.16b for the VM performance criterion, one can 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 4.15a and Figure 4.15b).

  Table 4.6 shows that EMLS-ONC-MO is never Pareto dominated for any type of instance. One can also notice that when the solutions proposed by EMLS-ONC-MO do not dominate the solutions proposed by the other approaches they still have an advantage. Indeed, as shown in the comparison of the average values of both EMLS-ONC-MO and the energy-aware FFD (-19 %, 1 %) respectively for energy savings and VMs performance, 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 of the scheduling 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. As a consequence, a shorter optimization time with less assignment possibilities is dedicated to the VM performance criterion.

  As for EMLS-ONC, Table 4.6 shows that with the two instances (320 hosts,

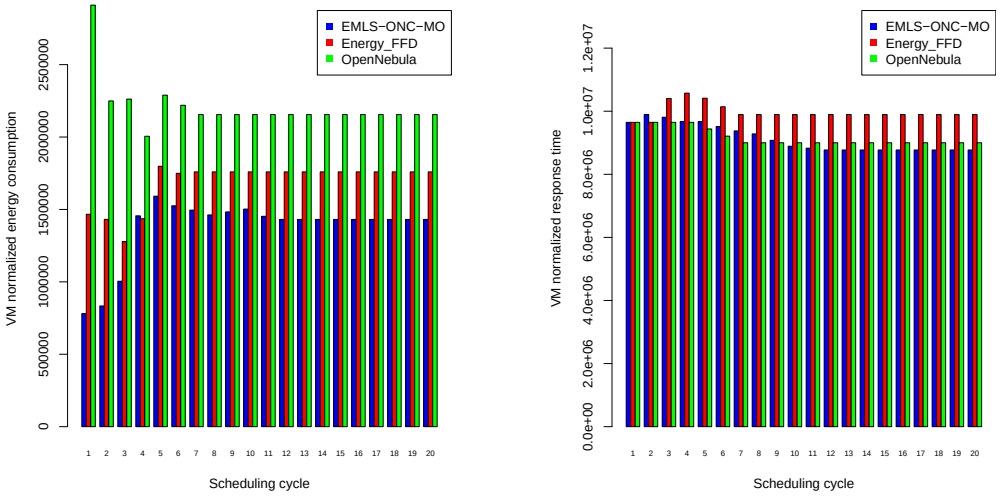
60 VMs) and (320 hosts, 100 VMs), EMLS-ONC-MO has worse results than energy-aware FFD in both objectives. However, this is not relevant because Table 4.6 reports none normalized values. Those values do not take into account the number of assigned VMs which is significantly higher for EMLS-ONC-MO. The normalized values drawn in Figure 4.16a and Figure 4.16b prove it.

Table 4.6: 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%** |

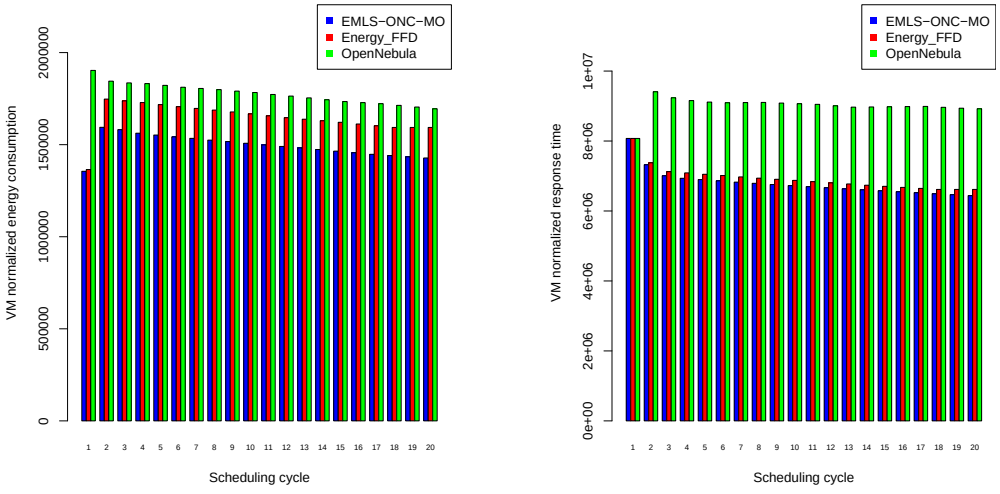
Table 4.7: Comparison between the processing time during 20 scheduling cycle in a row of the EMLS-ONC-MO, the energy-aware FFD and the OpenNebula scheduler.

| 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** |



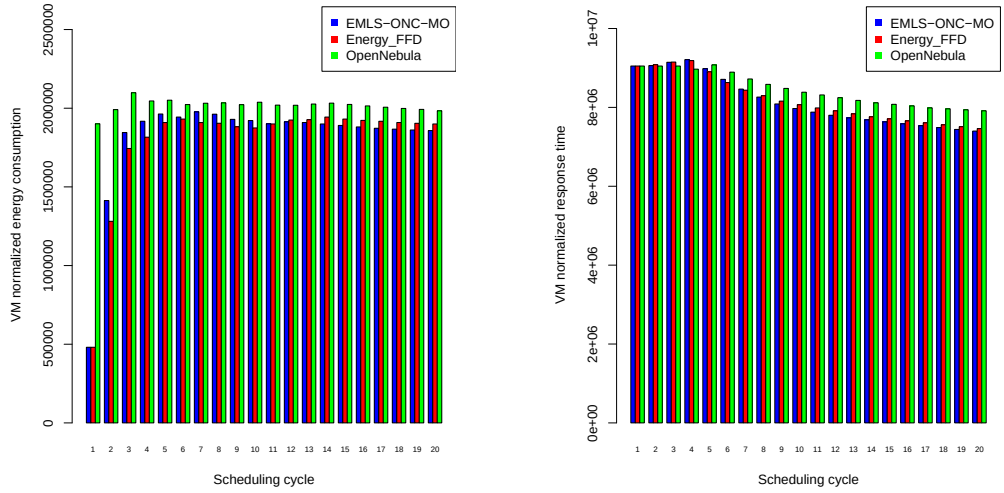

(a) (b)

Figure 4.14: Comparison of the results of (a): the normalized energy consumption. (b): 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.




(a) (b)

Figure 4.15: Comparison of the results of (a): the normalized energy consumption. (b): 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.




(a) (b)

Figure 4.16: Comparison of the results of (a): the normalized energy consumption. (b): 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.

#### 4.4.3.3 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.

- *Number of Scheduled VMs:* Table 4.8 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 is more significant against memory-aware FFD than against energy-aware FFD. However, EMLS-ONC-MO gives in average almost the same results as OpenNebula. This is due as said previously to the consolidation mechanism of the OpenNebula 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). Thus, we deduce that the 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 4.9 shows that the processing time of both the memory-aware FFD algorithm and the default

OpenNebula scheduler is significantly smaller than the scheduling cycle time. This is due as said before to the low complexity of the algorithms. We reported 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 terms of the performance of VMs (Figure 4.17b and Figure 4.19b). However, this efficiency has a drawback over the other criterion (i.e. the energy consumption) (see Figure 4.17a and Figure 4.19a). 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 4.17a) the less efficient EMLS-ONC-MO is regarding the VMs performances (see Figure 4.17b).

  Moreover, we notice that for instances with high VMs load such as in Figure 4.18a, 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 4.18b). One can see here the advantage of a Pareto EMLS-ONC-MO approach.

  Table 4.8 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 situations where the solutions proposed by EMLS-ONC-MO are dominated by the solutions of one of the other algorithms, are when EMLS-ONC-MO assigns more VMs than these algorithms. 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 is some cases 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 capacity has also a big number of CPUs

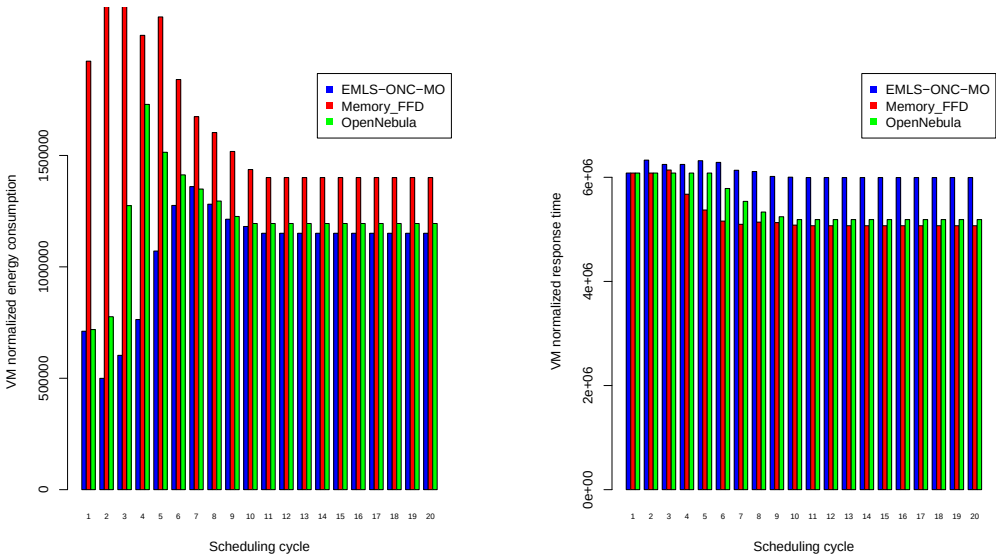
with a high frequency. Therefore, sorting the hosts by memory is equivalent to sorting them from the one with the best CPU capacity to the one with the worst. We deduce then that VM assignment using this technique consolidates the VMs in the hosts from the biggest to the smallest one. This consolidation is the reason why memory-aware FFD provides also good results in terms of energy.

Table 4.8: 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%** |

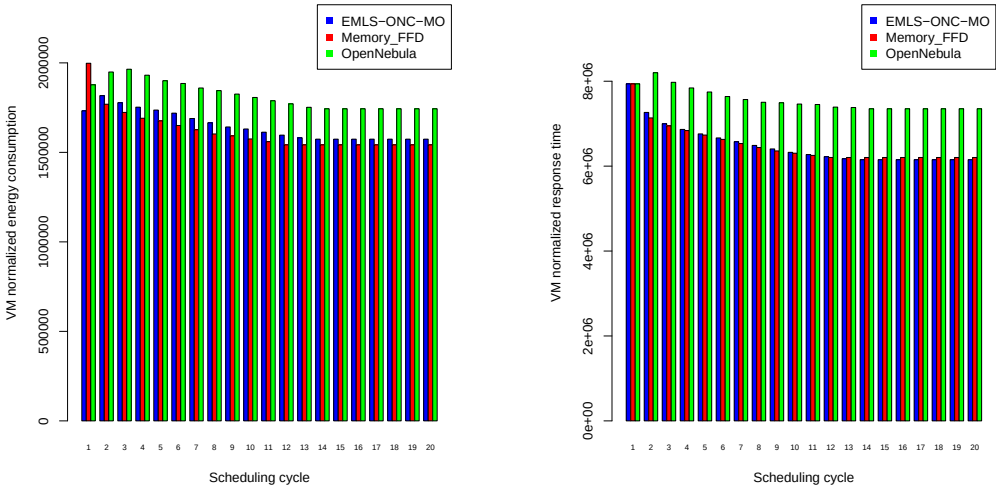
Table 4.9: Comparison between the processing time during 20 scheduling cycle in a row of the EMLS-ONC-MO, the memory-aware FFD and the OpenNebula scheduler.

| 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** |



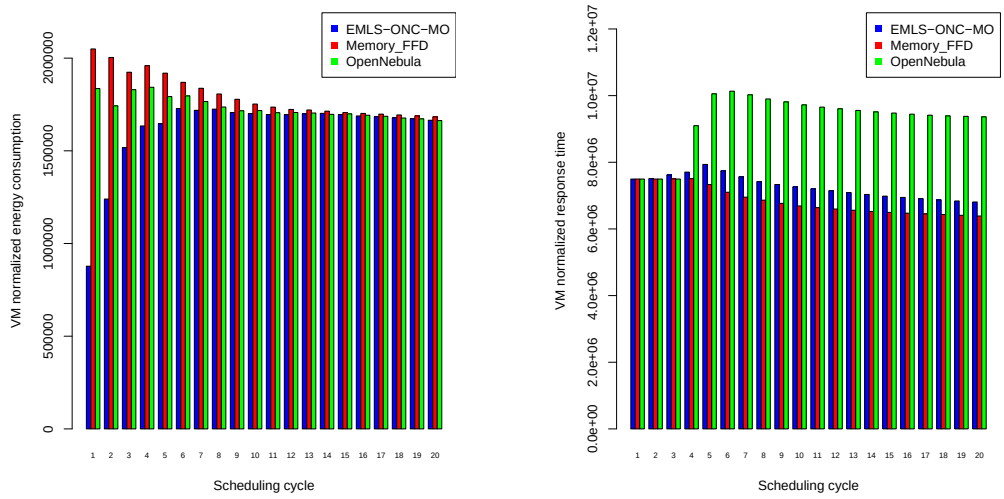

(a) (b)

Figure 4.17: Comparison of the results of (a): the normalized energy consumption. (b): 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.




(a) (b)

Figure 4.18: Comparison of the results of (a): the normalized energy consumption. (b): 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.




(a) (b)

Figure 4.19: Comparison of the results of (a): the normalized energy consumption. (b): 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.

#### 4.4.3.4 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 the GRID5000 infrastructure. The results are presented in Table 4.10 to Table 4.15. For each type of 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 4.10). However, for medium and high ratio instances, EMLS-ONC is better in the number of assigned VMs as presented in Table 4.12 and Table 4.14.

- *Computation time during each scheduling cycle:* the results show that EMLS-ONC never exceeds the scheduling cycle limit for any of the instances. For the instances where the scheduling time is set to 30 sec (i.e. (50 hosts, 5 VMs) and (50 hosts, 100 VMs)), the longest processing time registered is of 24.25$sec$

(see Table 4.12). Regarding the last instance, the scheduling cycle time is 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 and adds latency to retrieve the hosts' features. The obtained results show that EMLS-ONC spends for the longest processing time $53.68sec$ (see Table 4.14).

- *Energy consumption savings:* we can notice three different behaviors from the energy-based analysis of the experiment results 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 4.10 and Table 4.11).

  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 leaves 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 4.12). However, because of a higher number of assigned VMs, EMLS-ONC has better normalized energy values than the OpenNebula default scheduler (see Table 4.13).

  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 4.14 and Table 4.15). Even if EMLS-ONC assigns more VMs than the OpenNebula scheduler, this is not the cause of the performance decreasing.

  In the following, we will explain the cause of this performance drop on the instances with a medium and a high ratio. As shown in Figure 4.20, the policy of EMLS-ONC is to assign from the beginning the VMs in 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 4.20, two different assignments. The first at the right of Figure 4.20 (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 (4.2). Indeed, to be as precise as possible, our energy function evolves according to the previous CPU usage value of the host. Therefore, Equation (4.2) 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 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. Conversely, the default OpenNebula scheduler avoids to suffer from the $CPU\_usage_i$ parameter since each scheduling cycle brings a big number of VMs assigned using a consolidation policy, fully filling the host at once. To avoid this phenomenon, and to be more realistic about the energy consumption, the 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 real evolution of the energy consumption in a fair way regardless the specifications of the used policy.

In addition, another reason for the energy-aware performance drop of EMLS-ONC is the weak heterogeneity of the infrastructure that we used for the real experiments. The latter is composed of just two different types of machines (2 clusters). In contrast, this problem is less noticeable in the artificial experiments where the heterogeneity is more 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.

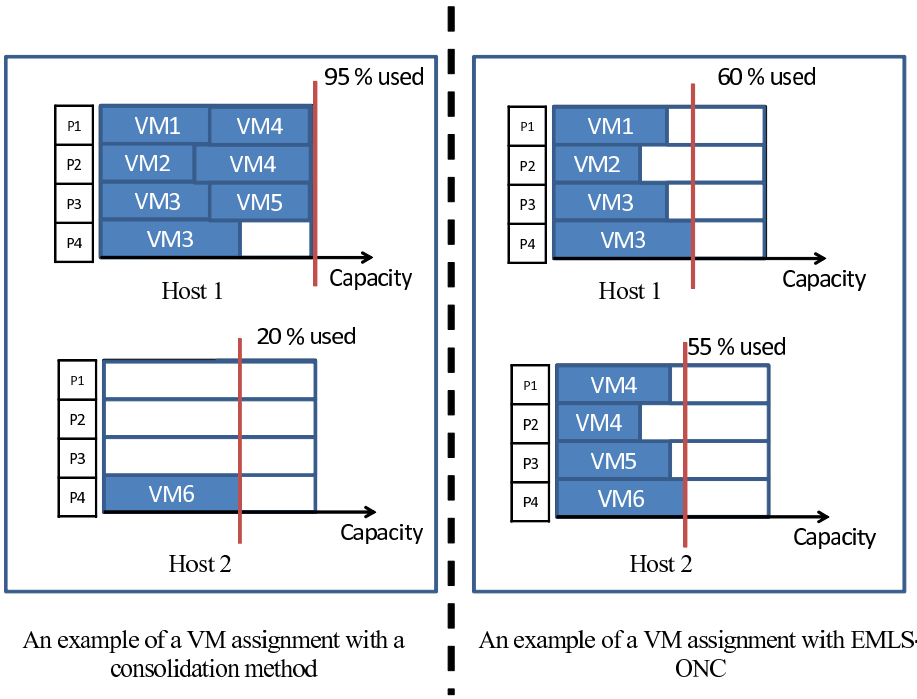



Figure 4.20: The difference between the assignment policies of EMLS-ONC and a consolidation based algorithm (OpenNebula scheduler) on a low heterogeneity architecture.

Table 4.10: Comparison raw results (EMLS-ONC vs OpenNebula scheduler) for 5 VMs/cycle on 50 GRID5000 machines.

| 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 4.11: Comparison cumulative results (EMLS-ONC vs OpenNebula scheduler) for 5 VMs/cycle on 50 GRID5000 machines.

| Type of instance | | Cumulative Energy Consumption | | Cumulative # VMs Assigned | | Normalized Energy Consumption | | EMLS-ONC vs OpenNebula |
|---|---|---|---|---|---|---|---|---|
| # Hosts | # VMs | EMLS-ONC | OpenNebula | EMLS-ONC | OpenNebula | EMLS-ONC | OpenNebula | Normalized Energy 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 4.12: Comparison raw results (EMLS-ONC vs OpenNebula scheduler) for 100 VMs/cycle on 50 GRID5000 machines.

| 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 | 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%** |

Table 4.13: Comparison cumulative results (EMLS-ONC vs OpenNebula scheduler) for 100 VMs/cycle on 50 GRID5000 machines.

| 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 4.14: Comparison of raw results (EMLS-ONC vs OpenNebula scheduler) for 100 VMs/cycle on 200 GRID5000 machines.

| 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 |
| 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%** |

Table 4.15: Comparison cumulative results (EMLS-ONC vs OpenNebula scheduler) for 100VMs/cycle on 200 GRID5000 machines.

| 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 |
| 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%** |

## 4.5 Conclusion

In this chapter, we have presented a new approach for the VM-level scheduling, the third and lowest level of scheduling in the cloud. This contribution has been integrated in a real cloud manager named OpenNebula using a multi-start local search metaheuristic. Two variants of this metaheuristic have been proposed, a single-objective one called EMLS-ONC and a Pareto multi-objective one named EMLS-ONC-MO. The main challenge at this level of scheduling is the minimization of the energy consumption of the managed infrastructure. However, addressing only the energy issue raises a new challenge of the VMs performance. In this purpose, optimizing only the energy criterion showed the advantage offered by EMLS-ONC, while minimizing both criteria using the Pareto EMLS-ONC-MO, led to find the tradeoff between both the energy consumption and the offered VMs performance. This is made possible by exploiting the heterogeneity offered by the different types of machines that compose the cloud infrastructures. The main contributions of this chapter are the following:

- *Cloud manager embedded scheduler:* we have shown that the lowest level of scheduling has an important impact on the results of both the service- and task-level scheduling if not optimized in terms of energy and performance. Therefore, the proposed scheduling techniques in this chapter have been embedded in a cloud manager to respect the real constraints related to a cloud infrastructure and to be as seamless as possible to be used by different upper configuration of the market-oriented clouds. We have chosen as a case study for managing our cloud the well known OpenNebula cloud manager.

- *New energy and performance models:* we have improved the models of the energy consumption of a host by relating the CPU usage to the increase of the energy consumption. Indeed, we have highlighted the relationship between the energy consumption of a host and its CPU usage value and gave the level of variation value that relates that CPU usage to the energy increase. Moreover, we have shown that the performance of the VMs is related to the memory usage of the host. In this purpose, we have proposed a VMs performance evaluation function based on the memory usage parameter.

- *Pareto EMLS-ONC-MO:* in addition to a single-objective energy-aware scheduler (EMLS-ONC), we have proposed a Pareto multi-objective scheduler that tackles also the VM performance objective. The idea was unlike other literature approaches, to relate both well known antagonist criteria of the scheduling on clouds by addressing them simultaneously. Indeed, each criterion taken separately impacts the other. Thus, the race to performance leads to a huge increase in the energy consumption and conversely, the reduction in the energy consumption decreases the cloud performance.

- *Real infrastructure deployment:* to fit the reality and demonstrate the feasibility of our contribution, we have proposed in addition to artificial experiments

over virtual clouds, a full analysis of real successful deployments of EMLS-ONC over up to 200 physical machines of the GRID'5000 grid infrastructure.

# Conclusion and Perspectives

The cloud computing paradigm is being more and more frequent in the daily users' life as an evolution of the IT. This concept helps the users through proposed services to satisfy their requests easily by suiting more precisely their needs. However, this generalization of cloud computing usage added different categories of users (clients) which led to different service levels. Indeed, depending on the accuracy of the needs or the business orientation of the service, different levels should be distinguished. Each cloud level brings its own specifications with its own issues. Thus, in this work we identified three different levels of clouds according to the service granularity: service-level, task-level and VM-level. For each cloud level we proposed a different metaheuristic-based algorithm to optimize the scheduling at this level. Two algorithms have been designed for a market-oriented cloud: MO-GA for the service-level scheduling addressing a wide computation-intensive service and MOGA-CB for the task-level scheduling dealing with a more accurate service in a brokering environment. The third algorithm is not market-oriented and concerns the common prerequisite of any cloud: the infrastructure management. It relies on a VM-level scheduling which addresses the efficiency related to the infrastructure as a service.

Several contributions related to each scheduling level have been presented in this thesis. For the service-level, we showed that high-level scheduling (meta-scheduling) has significant impact for addressing some issues such as energy consumption, profit and QoS. This is due to the geographical distribution nature of the meta-scheduled infrastructure. Indeed, the infrastructure that we deal with in the service level-scheduling is a cloud federation. The resulting high heterogeneity of such an infrastructure opens a lot of optimization possibilities using scheduling. Moreover, because of the number of criteria and their conflicting nature, we have proposed a multi-objective genetic algorithm (MO-GA) that faces all the issues at once to get a Pareto optimization that does not neglect any of them. As a metaheuristic, MO-GA successfully rose to this challenge by providing a wide range of solutions proven as efficient. Besides, the Pareto set of solutions permits to offer plenty of equivalent solutions regarding different criteria. However, to benefit from this diversity, we have proposed a selection mechanism based on a vector called meta-selection. It helps the provider to satisfy at any time the QoS of the client by acting directly over his/her criteria according to the current needs.

Regarding the task-level scheduling, the idea is conversely to the other approaches in the literature, to address the common impacting criteria of both the client's satisfaction and the broker's profit objectives instead of dealing directly with those objectives. Thus, we dealt simultaneously with the response time and the cost of the VM instances to address the satisfaction and profit issues. In

that way, we designed a dynamic Pareto evolutionary algorithm (MOGA-CB) to find the tradeoff between those criteria and to fit the scheduling constraints such as the respect of the scheduling round duration. Using our algorithm conducted to improve the usage of the utility theory in economics for modeling the client's satisfaction. The reported results showed that our Pareto-based approach helps to dispose from the client's requested preference parameters. That was made possible by providing a Pareto optimal or near-optimal solution giving a native tradeoff between the addressed criteria and selecting from the Pareto set the nearest solution to the origin of the axis.

Finally, the VM-level scheduling, as the common scheduling level to all the other scheduling levels of the cloud, plays a key role in the results of both the service- and task-level scheduling. Therefore, we embedded our proposed VM-level scheduler in the OpenNebula cloud manager to respect the real constraints related to a cloud infrastructure and to be as seamless as possible regarding the upper configuration of the market-oriented clouds. Moreover, we improved the energy consumption computation model of the hosts by relating the CPU usage value to the energy consumption. Also, we showed that the performance of the VMs was related to the memory usage of the host and proposed a new VMs performance evaluation function based on that observation. In addition, we considered together both energy and performance, the two best known antagonist criteria of a cloud infrastructure, by addressing them simultaneously using a Pareto multi-start local search algorithm. Finally, we conducted a full analysis of real successful cloud deployments of our VM-level scheduler over up to 200 physical machines of the Grid'5000 grid infrastructure.

This thesis opens a lot of perspectives related, on the one hand, to both the used models and the proposed approaches in our contributions and, on the other hand, to the combination of whole these contributions. The main model perspective in the first contribution, with the meta-scheduler for the service level-scheduling, is to determine a way to minimize more significantly the energy consumption by using DVS within each individual cloud of the federation, and to allow delays in the services' deadlines by introducing a new pricing model with penalties to enlarge the possible assignments. The major perspective of the second contribution lies in the usage of a simplified economic model dispossessed from the preference parameters ($\alpha$, $\beta$) to minimize with more impact both the response time and the price of the tasks. For the VM-level scheduling, the idea is to enhance the energy model by including other energy consumption sources like memory and/or hard drives to minimize with more impact the energy consumption. Moreover, a model that introduces a real time updating of the host parameter values is also interesting to have a more precise energy estimate.

Regarding the algorithm improvement perspectives, one can imagine a dynamic meta-scheduler in the service-level cloud federation which reassigns services during a new scheduling cycle on different clouds to improve the optimization of the

criteria. However, this is hard to implement in the real-cloud and strongly depends on the providers' policies. Indeed, the market competition forces the provider to adopt different frameworks to have exclusivity over their services and prevents the migration of their clients elsewhere. In the same way, based on the live migration process offered by the cloud managers (e.g. OpenNebula), a dynamic version of the EMLS-ONC/EMLS-ONC-MO scheduler for VM-level scheduling might be able to reassign the VMs during their running phase on different hosts for better optimization results. However, as for the service-level scheduling, this is hard to implement and depends on several issues related to the migrated VMs such as their flexibility (i.e the migration permissiveness of the running applications), their data transfer cost and their CPU time complexity.

Furthermore and as previously said, a more general perspective related to the whole contributions of this thesis can be proposed. It consists in the combination of all the different levels of scheduling as parts of the same scheduling workflow as shown in Figure 4.21. The result will be a general scheduling approach regardless the granularity of the service. One can see that a client can send a request to the cloud. Depending on the accuracy of the needs the request will either start by finding the convenient cloud among the federation using the meta-scheduler at the first level or directly be transmitted to the brokering scheduler at the second level. The next step at the second level will be the assignment of the client request to the judicious combination of the VM instances by the brokering scheduler. At the last level, the VM-level, the cloud manager scheduler will be in charge of finding for each VM proposed by the task-level, the best physical machine where it could be hosted efficiently.

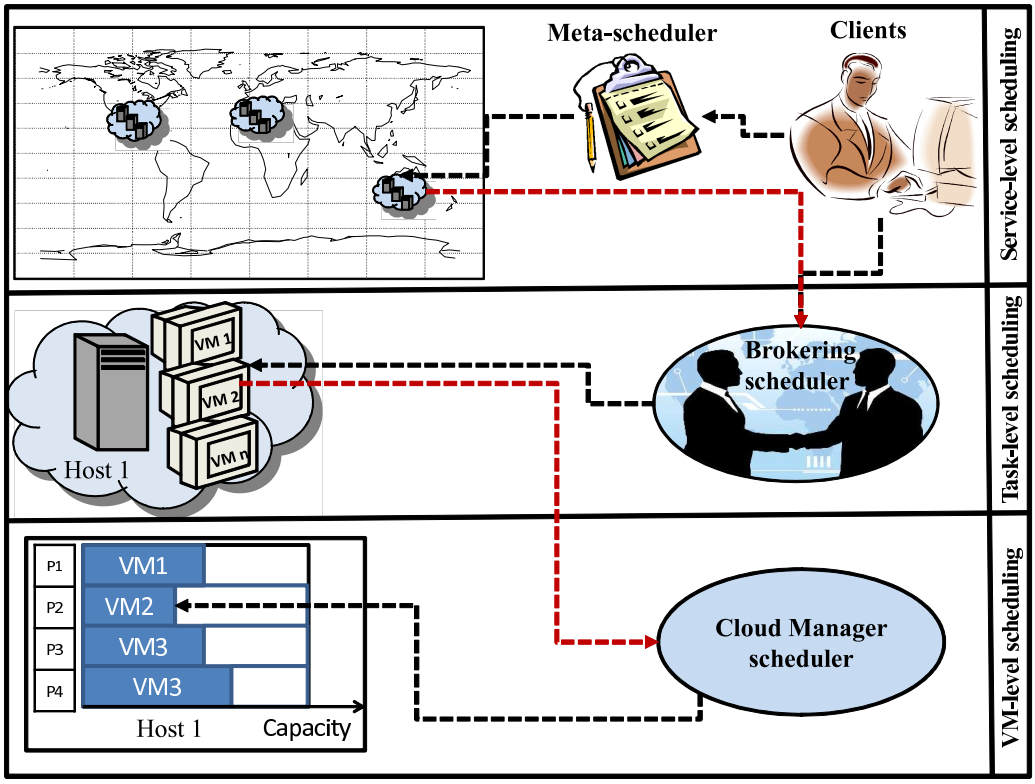



Figure 4.21: A General overview of the metaheurstics-based multi-level cloud scheduler. (Red arrows represent the information exchanges between the different levels of scheduling and black ones represent the communications at each level)

Appendix A

# Open Source Cloud Managers

Table A.1: Open source based cloud managers characteristics

| Feature | Eucalyptus | OpenNebula | Nimbus |
|---|---|---|---|
| Computing Architecture | -Ability to configure multiple clusters, each with private internal network addresses, into a single Cloud. - Private Cloud | -Cluster into an IaaS cloud- Focused on the efficient, dynamic and scalable management of VMs within datacenters (private clouds) involving a large amount of virtual and physical servers | -Science Cloud -Client-side cloud computing interface to Globus-enabled TeraPort cluster -Nimbus Context Broker that combines several deployed virtual machines into "turnkey" virtual clusters -Heterogeneous clusters of auto-configuring VMs with one command |
| Virtualization Management | -Xen hypervisor | -Xen, KVM and on-demande access to Amazon EC2 | -Xen Virtualization |
| Service | IaaS | IaaS | IaaS |
| Load balancing | -Simple load-balancing cloud controller | -Nginx Server configured as load balancer, used round-robin or weighted selection mechanism | -Launches self-configuring virtual clusters i.e. the context broker |
| Fault tolerance | -Separate cluster within the Eucalyptus cloud reduce the chance of correlated failure | -The deamon can be restarted and all the running VMs recovered -Persistent database backend to store host and VM information | -Checking worker nodes periodically and recovery |
| Interoperability | -Multiple cloud computing interfaces using the same "back end" infrastructure | -Interoperable between intra cloud services | -Standards: "rough consensus and working code" |
| Storage | -Walrus (the front end for the storage subsystem) | -Database, persistent storage for ONE data structures -SQLite3 backend is the core component of the OpenNebula internal data structures | GridFTP and SCP |
| Security | WS-security for authentication, Cloud controller generates the public/private key | -Firewall, Virtual Private Network Tunnel | PKI credential required - Works with Grid proxies VOMS, Shibboleth (via GridShib), custom PDPs |
| Programming Framework | -Hibernate, Axis2 and Axis2c, Java | -Java, Ruby, C++ | Python, Java |

Appendix B

# Preliminary Work for a Transition from the System-level Scheduling to the Cloud VM-level Scheduling

## B.1 Introduction

Precedence-constrained parallel applications are one of the most typical application model used in scientific and engineering fields. Such applications can be deployed on homogeneous or heterogeneous systems (HCSs) like cloud computing infrastructures.

Cloud computing is a simple concept that has emerged from heterogeneous distributed computing, grid computing, utility computing, and autonomic computing. In cloud computing, end-users do not own any part of the infrastructure. The end-users simply use the services available through the cloud computing paradigm and pay for the used services. The cloud computing paradigm can offer any conceivable form of services, such as computational resources for high performance computing applications, web services, social networking, and telecommunications services. Several criteria determine the quality of the provided service, the production cost of this service, and therefore the price paid by the end-user. The duration of this service (makespan) and the consumed energy are among of these criteria. The idea is to provide end-users a more flexible service that takes into account the duration of the service and the consumed energy. End-users could then find the right compromise between these two conflicting objectives to solve their precedence-constrained parallel applications.

The problem of finding the right compromise between the resolution time and the energy consumed of a precedence-constrained parallel application is a bi-objective optimization problem. The solution to this problem is a set of Pareto points. Pareto solutions are those for which improvement in one objective can only occur with the worsening of at least one other objective. Thus, instead of a unique solution to the problem, the solution to a bi-objective problem is a (possibly infinite) set of Pareto points. To the best of our knowledge, there is no research published in the literature to solve the above problem with a Pareto approach.

However, many works have focused on precedence-constrained parallel applications (e.g., [Darbha 1998], [Zomaya 1999], [Topcuouglu 2002] and [Lee 2008]). Most of these works propose algorithms to minimize the makespan. Only recently that some works are interested in minimizing the energy consumption (e.g., [Kim 2007a], [Bunde 2006], [Zhu 2003], [Zhu 2004], [Ge 2005] and [Rountree 2007]).

Using green IT techniques can significantly reduce an organization's and ultimately a country's carbon footprint. The UN bought 461 tons of carbon offsets to ensure that the September 2009 Summit on Climate Change in New York was carbon neutral. According to [Cameron 2010], if green IT techniques were implemented that reduced the energy use of 10,000 computers in West Virginia by 50%, the deployment would produce 33 times less carbon in one year than that produced by the summit. A recent study on power consumption by servers [Koomey 2008] shows that, in 2005, the power used by servers represented about 0.6% of total U.S. electricity consumption. That number 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 use for servers doubled over the period 2000 to 2005 in worldwide. The number of transistors integrated into today's Intel Itanium 2 processor reaches nearly 1 billion. If this rate continues, the heat (per square centimeter) produced by future Intel processors would exceed that of the surface of the sun [Koch 2005]. This implies the possibility of worsening system reliability, eventually resulting in poor system performance.

Due to the importance of energy consumption, various techniques including dynamic voltage scaling (DVS), resource hibernation, and memory optimizations have been investigated and developed [Venkatachalam 2005]. DVS among these has been proven to be a very promising technique with its demonstrated capability for energy savings (e.g., [Bunde 2006], [Zhu 2003] and [Ge 2005]). For this reason, we adopt this technique and it is of particular interest to this study. DVS enables processors to dynamically adjust voltage supply levels (VSLs) aiming to reduce power consumption. However, this reduction is achieved at the expense of sacrificing clock frequencies.

In this paper, we investigate the energy issue in task scheduling particularly on HCSs like cloud computing systems. We propose a new parallel bi-objective hybrid genetic algorithm that takes into account, not only makespan, but also energy consumption. Our new approach is a hybrid between a multi-objective parallel genetic algorithm and energy-conscious scheduling heuristic (ECS) [Lee 2009]. The results clearly demonstrate the superior performance of ECS over the other algorithms like DBUS [Bozdag 2006] and HEFT [Topcuouglu 2002]. Genetic algorithms make it possible to explore a great range of potential solutions to a problem. The exploration capability of the genetic algorithm and the intensification power of ECS are complementary. A skillful combination of a metaheuristic with concepts originating from other types of algorithms lead to more efficient behavior.

Our algorithm is effective as it profits from the exploration power of the genetic algorithm, the intensification capability of ECS, the cooperative approach of the island model, and the parallelism of the multi-start model. The island model and the hybridization improve the quality of the obtained results. The multi-start model reduces the running time of a resolution. Furthermore, one of the major interests of our approach is to give the end-user a set of Pareto solutions to choose according to the desired quality of service, in particular the completion time, and the cost

of the service in terms of energy and consequently in terms of price willing to pay. The proposed method can easily be applied to loosely coupled HCSs (e.g., cloud computing systems) using advance reservation and various sets of frequency-voltage pairs.

Our new approach is evaluated with the Fast Fourier Transformation task graph which is a real-world application. Experiments show that (1) the hybridization improves on average the best known results obtained in the literature (by **47.5%** for the energy consumption and **12%** for the completion time), (2) the island model significantly improves the results obtained using only the hybridization, and (3) the multi-start model accelerates our approach with an average speedup of **13** using 21 cores.

The remainder of the paper is organized as follows. Section B.2 presents the application, system, energy and scheduling models used in this paper. Section B.3 describes the related work. Our algorithm is presented in Section B.4. The results of our comparative experimental study are discussed in Section B.5. The conclusion is drawn in Section B.6. The paper ends with an appendix which describes our approaches using pseudo-code.

## B.2 Problem modeling

In this section, we describe the system, application, energy and scheduling models used in our study.

### B.2.1 Cloud computing model

A cloud computing system is a set of resources designed to be allocated ad hoc to run applications. In our model, the cloud is assumed to be hosted in a data center which is composed by heterogeneous machines. This data center provides a set of services hosted on thousands of high-end computing servers. The need in terms of services of an application can be modeled by a task graph. In this graph, an edge between two tasks represents an inter-service communication.

The cloud computing system used in this work consists of a set $P$ of $p$ heterogeneous processors/machines. Each processor $p_j \in P$ is DVS-enabled; in other words, it can operate with different VSLs (i.e., different clock frequencies). For each processor $p_j \in P$, a set $V_j$ of $v$ VSLs is random and uniformly distributed among three different sets of VSLs (Table B.1). Since clock frequency transition overheads take a negligible amount of time (e.g., $10\mu s$- $150\mu s$ [Intel 2004], [Min 2000]), these overheads are not considered in our study. The inter-processor communications are assumed to perform with the same speed on all links without contentions. It is also assumed that a message can be transmitted from one processor to another while a task is being executed on the recipient processor which is possible in many systems.

Table B.1: Voltage-relative speed pairs

| | Pair 1 | | Pair 2 | | Pair 3 | |
|---|---|---|---|---|---|---|
| Level | Voltage ($v_k$) | Relative speed (%) | Voltage ($v_k$) | Relative speed (%) | Voltage ($v_k$) | Relative speed (%) |
| 0 | 1.5 | 100 | 2.2 | 100 | 1.75 | 100 |
| 1 | 1.4 | 90 | 1.9 | 85 | 1.4 | 80 |
| 2 | 1.3 | 80 | 1.6 | 65 | 1.2 | 60 |
| 3 | 1.2 | 70 | 1.3 | 50 | 0.9 | 40 |
| 4 | 1.1 | 60 | 1.0 | 35 | | |
| 5 | 1.0 | 50 | | | | |
| 6 | 0.9 | 40 | | | | |

### B.2.2 Application model

Parallel programs can be generally represented by a directed acyclic graph (DAG). A DAG, $G = (N, E)$, consists of a set $N$ of $n$ nodes and a set $E$ of $e$ edges. A DAG is also called a task graph or macro-dataflow graph. In general, the nodes represent tasks partitioned from an application; the edges represent precedence constraints. An edge $(i, j) \in E$ between task $n_i$ and task $n_j$ also represents inter-task communication. A task with no predecessors is called an entry task, $n_{entry}$, whereas an exit task, $n_{exit}$, is one that does not have any successors. Among the predecessors of a task $n_i$, the predecessor which completes the communication at the latest time is called the most influential parent (MIP) of the task denoted as $MIP(n_i)$. The longest path of a task graph is the critical path.

The weight on a task $n_i$ denoted as $w_i$ represents the computation cost of the task. In addition, the computation cost of the task on a processor $p_j$, is denoted as $w_{i,j}$ and its average computation cost is denoted as $\bar{w}_i$.

The weight on an edge, denoted as $c_{i,j}$ represents the communication cost between two tasks, $n_i$ and $n_j$. However, a communication cost is only required when two tasks are assigned to different processors. In other words, the communication cost when tasks are assigned to the same processor is zero and thus can be ignored.

The earliest start time of, and the earliest finish time of, a task $n_i$ on a processor $p_j$ is defined as

$$EST(n_i, p_j) = \begin{cases} 0 & if n_i = n_{entry} \\ EFT(MIP(n_i), p_k) + c_{MIP(n_i),i} & otherwise \end{cases}$$

$$EFT(n_i, p_j) = EST(n_i, p_j) + w_{i,j}$$

Note that the actual start and finish times of a task $n_i$ on a processor $p_j$, denoted as $AST(n_i, p_j)$ and $AFT(n_i, p_j)$ can be different from its earliest start and finish times, $EST(n_i, p_j)$ and $EFT(n_i, p_j)$, if the actual finish time of another task scheduled on the same processor is later than $EST(n_i, p_j)$.

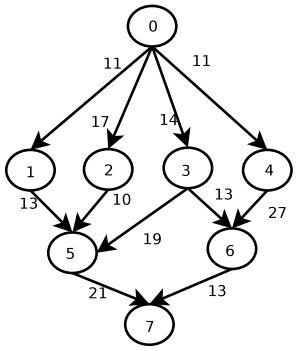



Figure B.1: A simple task graph

In the case of adopting task insertion the task can be scheduled in the idle time slot between two consecutive tasks already assigned to the processor as long as no violation of precedence constraints is made. This insertion scheme would contribute in particular to increasing processor utilization for a communication intensive task graph with fine-grain tasks.

A simple task graph is shown in Figure B.1 with its details in Table B.2 and Table B.3. The values presented in Table B.3 are computed using two frequently used task prioritization methods, t-level and b-level. Note that, both computation and communication costs are averaged over all nodes and links. The t-level of a task is defined as the summation of the computation and communication costs along the longest path of the node from the entry task in the task graph. The task itself is excluded from the computation. In contrast, the b-level of a task is computed by adding the computation and communication costs along the longest path of the task from the exit task in the task graph (including the task). The b-level is used in this study.

The communication to computation ratio (CCR) is a measure that indicates whether a task graph is communication intensive, computation intensive or moderate. For a given task graph, it is computed by the average communication cost divided by the average computation cost on a target system.

### B.2.3 Energy model

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 to be 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 capacitive power ($P_c$) is defined as

$$P_c = ACV^2 f, \tag{B.1}$$

where $A$ is the number of switches per clock cycle, $C$ is the total capacitance load, $V$ is the supply voltage, and $f$ is the frequency. Equation (B.1) clearly indicates that the supply voltage is the dominant factor; therefore, its reduction would be most

Table B.2: Computation cost with VSL 0

| Task | $p_0$ | $p_1$ | $p_2$ |
|---|---|---|---|
| 0 | 11 | 13 | 9 |
| 1 | 10 | 15 | 11 |
| 2 | 9 | 12 | 14 |
| 3 | 11 | 16 | 10 |
| 4 | 15 | 11 | 19 |
| 5 | 12 | 9 | 5 |
| 6 | 10 | 14 | 13 |
| 7 | 11 | 15 | 10 |

Table B.3: Task Priorities

| Task | b-level | t-level |
|---|---|---|
| 0 | 101.33 | 0.00 |
| 1 | 66.67 | 22.00 |
| 2 | 63.33 | 28.00 |
| 3 | 73.00 | 25.00 |
| 4 | 79.33 | 22.00 |
| 5 | 41.67 | 56.33 |
| 6 | 37.33 | 64.00 |
| 7 | 12.00 | 89.33 |

influential to lower power consumption. The energy consumption of the execution of a precedence-constrained parallel application used in this study is defined as

$$E = \sum_{i=0}^{n} ACV_i^2 f.w_i^* = \sum_{i=0}^{n} \alpha V_i^2 w_i^*,$$

where $V_i$ is the supply voltage of the processor on which task $n_i$ is executed, and $w_i^*$ is the computation cost of task $n_i$ (the amount of time taken for $n_i$'s execution) on the scheduled processor.

### B.2.4 Scheduling model

The task scheduling problem in this study is the process of allocating a set $N$ of $n$ tasks to a set $P$ of $p$ processors (without violating precedence constraints) that minimizes makespan with energy consumption as low as possible. The makespan is defined as $M = max\{AFT(n_{exit})\}$ after the scheduling of $n$ tasks in a task graph $G$ is completed. Although the minimization of makespan is crucial, tasks of a DAG in our study are not associated with deadlines as in real-time systems.

## B.3 Related work

In this section, we present some noteworthy works in task scheduling, particularly for HCSs, and then scheduling algorithms with power/energy consciousness.

### B.3.1 Scheduling in HCSs

Due to the NP-hard nature of the task scheduling problem in general cases [Garey 1979], heuristics, in particular meta-heuristics, are the most popularly adopted scheduling approaches. List scheduling heuristics are the dominant heuristic technique. This is because empirically, list scheduling algorithms tend to produce competitive solutions with lower time complexity compared to algorithms in the other categories [Kwok 1998].

The HEFT algorithm [Topcuouglu 2002] is highly competitive in that it generates a schedule length comparable to other scheduling algorithms, with a low time complexity ($O(nlogn + (e + n)p)$). It is a list-scheduling heuristic consisting of the two typical phases of list scheduling (i.e., task prioritization and processor selection) with task insertion.

The DBUS algorithm [Bozdag 2006] is a duplication-based scheduling heuristic that first performs a critical path based listing for tasks and schedules them with both task duplication and insertion. The experimental results in [Bozdag 2006] show its attractive performance, especially for communication-intensive task graphs. The time complexity of DBUS is in the order of $O(n^2p^2)$.

### B.3.2 Scheduling with energy consciousness

Most of previous studies on scheduling with the consideration of energy consumption are conducted on homogeneous computing systems [Kim 2007a], [Zhu 2003], [Zhu 2004], [Ge 2005], [Rountree 2007], [Chen 2005a] or single-processor systems [Zhong 2007]. In addition to system homogeneity, tasks are generally homogeneous and independent. Slack management/reclamation is a frequently adopted technique with DVS.

In [Zhu 2003], several different scheduling algorithms using the concept of slack sharing among DVS-enabled processors were proposed. The rationale behind the algorithms is to utilize idle (slack) time slots of processors lowering supply voltage (frequency/speed). This technique is known as slack reclamation. These slack time slots occur, due to earlier completion (than worst case execution time) and/or dependencies of tasks. The work in [Zhu 2003] has been extended in [Zhu 2004] with AND/OR model applications. Since the target system for both works is shared-memory multiprocessor systems, communication between dependent tasks is not considered unlike our approach.

In [Ge 2005], two voltage scaling algorithms for periodic, sporadic, aperiodic tasks on a dynamic priority single-processor system are proposed. They are more practical compared with many existing DVS algorithms in that *a priori* information

on incoming tasks is not assumed to be available until the tasks are actually released. This assumption does not correspond to our task model as explained.

*Rountree et al.* in [Rountree 2007] developed a system based on linear programming that exploits slack using DVS (i.e., slack reclamation). Their linear programming system aims to deliver near-optimal schedules that tightly bound optimal solutions. It incorporated allowable time delays, communication slack, and memory pressure into its scheduling. The linear programming system mainly deals with energy reduction for a given pre-generated schedule with a makespan constraint as in most existing algorithms. In our apprche, the makespan is not a constraint but an objective to optimize.

Another two scheduling algorithms for bag-of-tasks applications on clusters are proposed in [Kim 2007a]. Tasks in a bag-of-tasks application are typically independent and homogeneous, yet run with different input parameters/files. In [Kim 2007a], deadline constraints are associated with tasks for the purpose of quality control. The two algorithms differ in terms of whether processors in a given computer cluster are time-shared or space-shared. Computer clusters in this paper are composed of homogeneous DVS-enabled processors unlike our approach where processors are heterogeneous.

In [Wang 2010], the authors propose a formulation of energy aware scheduling algorithm and a detailed discussion of slack time computation. This scheduling algorithm also concerns reducing voltages during the communication phases between parallel jobs. In [Khan 2009], the authors study the energy-aware task allocation problem for assigning a set of tasks onto the machines of a computational grid each equipped with DVS feature. The goal is to optimize the energy consumption and response time in computational grids. Unlike our approach, [Khan 2009] suppose that tasks are independent and are not subject to precedence constraints.

To the best of our knowledge, none of previous scheduling approaches explicitly addresses the energy issue with a multi-objective approach when tackling the problem of scheduling precedence-constrained parallel applications on HCSs. Therefore, the scheduling algorithms with energy consciousness presented in this section are the most closely related works to our study.

### B.3.3 Energy-conscious scheduling heuristic

The consideration of energy consumption in task scheduling adds another layer of complexity to an already intricate problem. Unlike real-time systems, applications in our study are not deadline-constrained. Therefore, the evaluation of the quality of schedules should be measured explicitly considering both makespan and energy consumption. For this reason, energy-conscious scheduling heuristic (ECS) [Lee 2009] is devised with relative superiority (RS) as a novel objective function, which takes into account these two performance criteria.

For a given ready task, its RS value on each processor is computed using the current best combination of processor and VSL (p' and v' are, respectively, the selected processor and its voltage supply level) for that task, and then the processor

from which the maximum RS value is obtained is selected. Since each scheduling decision that ECS makes tends to be confined into a local optimum, another energy reduction technique (MCER) is incorporated into the energy reduction phase of ECS without sacrificing time complexity. It is an effective technique in lowering energy consumption, although the technique may not help schedules escape from local optima. MCER is makespan conservative in that changes it makes (to the schedule generated in the scheduling phase) are only validated if they do not increase the makespan of the schedule. For each task in a DAG, MCER considers all of the other combinations of task, host and VSL to check whether any of these combinations reduces the energy consumption of the task without increasing the current makespan.

The results clearly demonstrate the superior performance of ECS over DBUS and HEFT. Note that, in many previous studies [Kim 2007b], [Lee 2005], HEFT has been proven to perform very competitively, and it has been frequently adopted and extended.

However, ECS returns one solution as a result, and precedence-constrained applications problem is bi-objective in nature. Section B.4 presents our new parallel bi-objective approach based on hybridization between a genetic algorithm and ECS. This approach provides a set of solutions to this problem. The experiments presented in Section B.5 show that our approach often gives solutions which are better than those found by ECS.

## B.4 A parallel hybrid approach

This section starts with a brief overview on multi-objective combinatorial optimization and genetic algorithms. Then, our new parallel bi-objective hybrid approach is presented.

### B.4.1 Genetic algorithms

Genetic Algorithms (GAs) are meta-heuristics based on the iterative application of stochastic operators on a population of candidate solutions. At each iteration, solutions are selected from the population. The selected solutions are recombined in order to generate new ones. The new solutions replace other solutions selected either randomly or according to a selection strategy.

In the Pareto-oriented multi-objective context [Deb 2001], the structure of the GA remains almost the same as in the mono-objective context. However, some adaptations are required mainly for the evaluation and selection operators.

The selection process is often based on two major mechanisms: *elitism* and *sharing*. They allow respectively the convergence of the evolution process to the best Pareto front and to maintain some diversity of the potential solutions. The elitism mechanism makes use of a second population called a *Pareto archive* that stores the different non-dominated solutions generated through the generations. 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 selected either from the Pareto archive, from the population or from both of them. The sharing operator maintains the diversity on the basis of the similarity degree of each individual compared to the others. The similarity is often defined as the Euclidean distance in the objective space.

### B.4.2 Hybrid approach

In our approach illustrated in Figure B.2, a solution (chromosome) is composed of a sequence of $N$ genes. The $i^{th}$ gene of a solution s is denoted $s_j$. Each gene is defined by a task, a processor and a voltage. These three parts of $s_j$ are denoted respectively $t(s_j)$, $p(s_j)$ and $v(s_j)$. This means that the task $t(s_j)$ is assigned to the processor $p(s_j)$ with the voltage $v(s_j)$.

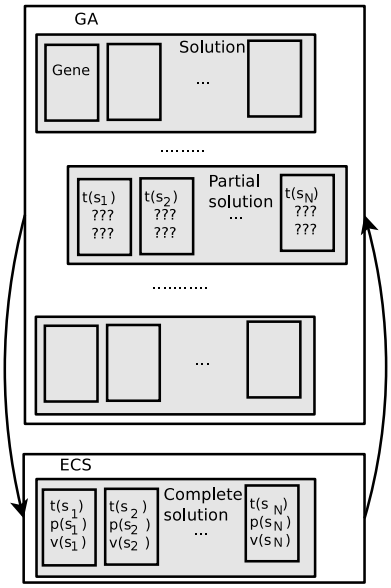



Figure B.2: Our hybrid GA (GA and ECS)

The new approach we propose is based on ECS [Lee 2009] which is not a population-based heuristic. ECS tries to construct in a greedy way one solution using three components.

- The first component to build the task parts of each gene of the solution.
- The second component to build the processor and voltage parts of these genes.
- And the third component to calculate the fitness of a solution in terms of energy consumption and makespan.

Unlike ECS, our approach provides a set of Pareto solutions. This approach is a hybrid between a multi-objective GA and the second component of ECS. The role of the GA is to provide good task scheduling. In other words, the GA builds task

parts $t(s_1)$, $t(s_2)$, ..., $t(s_n)$ of a solution $s$. Therefore, the mutation and crossover operators of the GA affect only the task part of the genes of each solution.

The second component of ECS is called whenever a solution is modified by these two operators. The first role of this component is to correct the task order to take into account the precedence constraints in the task graph. Then the component completes the processor and voltage parts of the genes of the partial solutions provided by these operators. In other words, ECS builds the remaining parts $p(s_1)$, $p(s_2)$, ..., $p(s_n)$ and $v(s_1)$, $v(s_2)$, ..., $v(s_n)$ of the partial solutions provided by the mutation and crossover operators of the GA.

The evaluation (fitness) operator of the GA is called once the task, processor and voltage parts of each gene of the solution are known. The role of this operator is to calculate the energy consumption and the makespan of each solution.

The mutation operator is based on the first component of ECS. This first component returns all tasks scheduled according to their b-level values. The principle of our mutation operator is also based on the scheduling of tasks according to their b-level values. The b-level concept is explained in Section B.2. It should be noted that one can choose the t-level values instead of those of b-level. First, the operator chooses randomly two integers $i$ and $j$ such that $1 \leq i < j \leq n$ and $b-level(t(s_i)) < b-level(t(s_j))$. Then, the operator swaps the two tasks $t(s_i)$ and $t(s_j)$ (see Figure B.3).

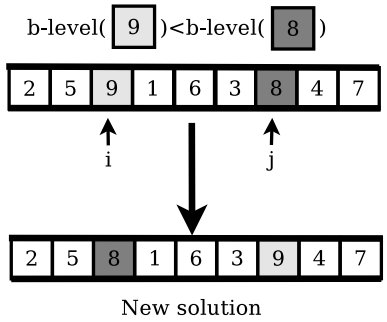



Figure B.3: The mutation operator

As illustrated in Figure B.4, the crossover operator uses two solutions $s1$ and $s2$ to generate two new solutions $s'1$ and $s'2$. To generate $s'1$, 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 $s'1$ all tasks of $s1$ located before $i$ or after $j$. These tasks are copied according to their positions ($s'1_k = s1_k$ if $k < i$ or $k > j$).
- copies in a solution $s$ all tasks of $s2$ that are not yet in $s'1$. 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 $s'1$ located between $i$ and $j$ ($s'1_k = s_{k-i+1}$ for all $i \leq k \leq j$).

The solution $s'2$ is generated with the same method by considering $s2$ as the first parent and $s1$ as the second parent.

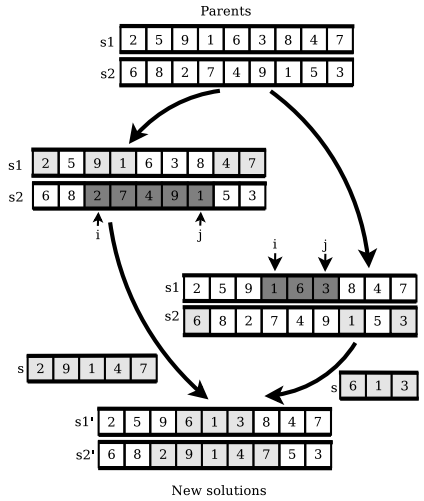



Figure B.4: The crossover operator

The other elements of the GA in the new approach are conventional. Indeed, our GA randomly generates the initial population. Its selection operator is based on a tournament strategy. The algorithm stops when no new best solution is found after a fixed number of generations.

### B.4.3 Insular approach

The island model [Cohoon 1987] is inspired by behaviors observed in the ecological niches. In this model, several evolutionary algorithms are deployed to evolve simultaneously various populations of solutions, often called islands. As shown in Figure B.5, the GAs of our hybrid approach asynchronously exchange solutions. This exchange aims at delaying the convergence of the evolutionary process and to explore more zones in the solution space. For each island, a migration operator intervenes at the end of each generation. Its role consists to decide the appropriateness of operating a migration, to select the population sender of immigrants or the receiver of emigrants, to choose the emigrating solutions, and to integrate the immigrant ones.

### B.4.4 Multi-start approach

Compared to the GA, ECS is more costly in CPU time. The different evaluations of ECS are independent of each other. Therefore, their parallel execution can make the approach faster. The objective of the hybrid approach is to improve the quality of solutions. The island approach also aims to obtain solutions of better quality. The

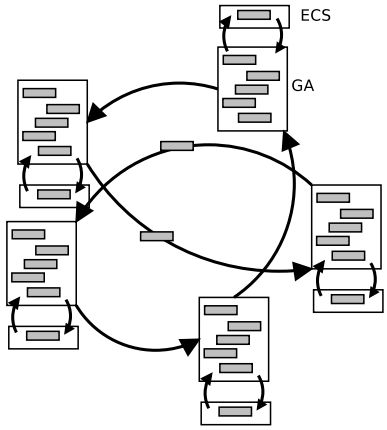



Figure B.5: The cooperative island approach

goal of the parallel multi-start approach is to reduce the execution time. As shown in Figure B.6, our parallelization is based on the deployment of the approach using the farmer-worker paradigm. The GA processes are farmers and ECS processes are workers.

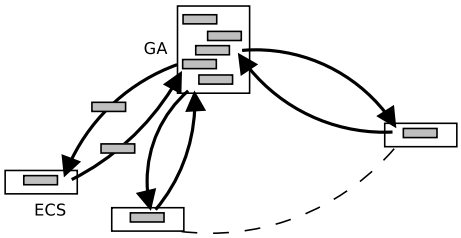



Figure B.6: Illustration of the multi-start approach

## B.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 hybridization, the insular approach and the multi-start approach respectively compared to ECS, the hybrid approach and the insular approach.

### B.5.1 Experimental settings

The performance of our approach was thoroughly evaluated with the Fast Fourier Transformation [Cormen 1990] task graph which is a real-world application. A large number of variations were made on this task graph for more comprehensive experiments. Various different characteristics of processors were also applied. Table B.4 summarizes the parameters used in our experiments.

Table B.4: Experimental parameters

| Parameter | Value |
|---|---|
| The number of tasks | ∼20 ∼40 ∼60 ∼80 ∼120 |
| The number of processors | 02 04 08 16 32 64 |
| Processor heterogeneity | 100 200 random |
| CCR | 0.1 0.2 1.0 5.0 10.0 |

The new approach is experimented on 9,000 instances distributed equitably according to the number of tasks, the number of processors, the processor heterogeneity and the CCR (1/5 of instances have a number of tasks equal to ∼20, 1/5 of instances have a number of tasks equal to ∼40,..., 1/6 of instances have a number of processors equal to 2, etc.). In other words, 20 instances are used for each combination of parameters.

Our approach has been implemented using ParadisEO [Cahon 2004]. This software platform provides tools for the design of parallel meta-heuristics for multi-objective optimization. [Talbi 2009] explains how to implement a multi-objective genetic algorithm, an insular approach, and a multi-start using ParadisEO. Table B.5 shows the parameters used by ParadisEO for the hybrid, insular and multi-start approaches during our experiments.

Table B.5: The parameters used by ParadisEO for each approach

| Parameters | Hybrid | Insular | Multi-start |
|---|---|---|---|
| Population size | 20 | 20 | 20 |
| Number of generations | 1000 | 100 | 100 |
| Crossover rate | 1 | 1 | 1 |
| Mutation rate | 0.35 | 0.35 | 0.35 |
| Migration topology | | Ring | |
| Migration rate | | Every 20 generations | |
| Number of migrants | | 5 | |

Experiments have been performed on a grid of three clusters. The first cluster contains 8 Opteron 244 nodes (dual-processor clocked at 1.8 GHz, 2 GB of RAM). The second contains 10 Xeon L5420 nodes (bi-quad-core processors clocked at 2.5 GHz, 16 GB of RAM). The third cluster contains 106 cores AMD opteron 248, 40 cores AMD opteron 252, 104 cores AMD opteron 285, et 368 cores Intel Xeon E5440 QC. A total of 714 cores are used. The first two clusters are located at the University of Mons in Belgium, while the third cluster is at Université de Lille1 in France.

### B.5.2 Hybrid approach

The hybrid approach is experimented on all instances of Table B.4. Each instance is solved twice. The first resolution is done with ECS, and the second resolution with the new approach. These experiments are launched by a script on one of the cores of our grid according to their availability.

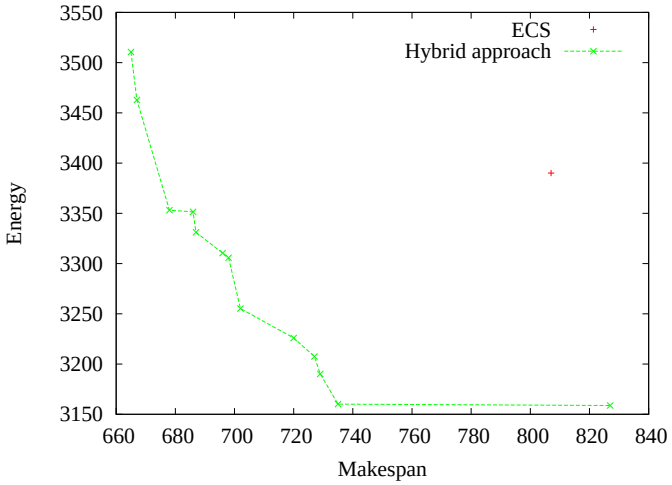



Figure B.7: An example of the obtained results with the hybrid approach and ECS for the same instance

Figure B.7 gives an example of a Pareto front obtained with the hybrid approach and the solution obtained by ECS for the same instance which is the tenth instance generated with the number of tasks equal to ∼20, number of processors equal to 02, processor heterogeneity equal to 100, and CCR equal to 0.1. Experiments show that ECS finds the solution of an instance after $\sim 1$ second on average, while our hybrid approach requires about $\sim 25$ minutes on average to find the Pareto solutions of an instance. As previously mentioned, ECS is a heuristic that builds one and only one solution using a greedy strategy, and our hybrid approach is based on the hybridization between a GA and ECS. Therefore, the hybrid approach handles a population of solutions that evolves over several generations, and the second component of ECS is called and used during the construction of each solution. So it was expected that the hybrid approach uses more computing power than ECS. Our goal is not to have an approach faster than ECS but an approach that gives Pareto solutions which improve the solution of ECS. Our approach is useful, for example, for large scientific applications requiring high computing power, and for small applications which are executed many times.

Table B.6 compares the Pareto solutions of the hybrid approach with the solution of ECS. The comparison is made according to the number of tasks, the number of processors, the processor heterogeneity and the CCR. The third column shows the average number of obtained Pareto solutions. The last column gives the percentage of Pareto solutions that improves the ECS solution on the two objectives

simultaneously. As indicated in the last line of the table, the hybrid approach provided **19.77** solutions on average, and **83.04%** of the Pareto solutions found by this hybrid approach improve the ECS solution on the two objectives simultaneously. In addition, Table B.6 shows that the more tasks there are, the more Pareto solutions are found, and the more the percentage of Pareto solutions dominating the ECS solution increases.

Table B.6: Comparison of Pareto hybrid approach solutions and ECS solution

| | | Average number of Pareto solutions | Pareto solutions dominating ECS solution (%) |
|---|---|---|---|
| Number of tasks | ∼20 | 14.78 | 78.24 |
| | ∼40 | 19.57 | 80.70 |
| | ∼60 | 21.36 | 83.62 |
| | ∼80 | 21.45 | 83.12 |
| | ∼120 | 21.67 | 89.51 |
| Number of processors | 02 | 18.51 | 73.21 |
| | 04 | 19.42 | 71.01 |
| | 08 | 22.17 | 75.83 |
| | 16 | 23.32 | 86.12 |
| | 32 | 19.98 | 94.73 |
| | 64 | 15.18 | 97.36 |
| Heterogeneity | 100 | 20.12 | 80.82 |
| | 200 | 21.67 | 74.47 |
| | random | 17.52 | 93.83 |
| CCR | 0.1 | 19.60 | 88.44 |
| | 0.2 | 19.47 | 88.83 |
| | 1.0 | 17.53 | 89.09 |
| | 5.0 | 19.26 | 78.80 |
| **Average** | | **19.77** | **83.04** |

To determine the contribution of the new approach, in terms of the values of makespan and energy consumption, we compare the solution provided by ECS to only one solution of the Pareto set provided by the new approach. The solution chosen in the Pareto set is used only to compare the new approach with ECS. Nevertheless the decision maker, using the new approach, will have a set of Pareto solutions instead of one solution.

For each instance,

- a first resolution is done using ECS to provide one solution $s$.
- a second resolution is done using the new approach to obtain a set $E$ of Pareto solutions.

- only one Pareto solution $s'$ is selected from the set $E$. This solution is the closest to $s$ in the sense of Euclidean distance.
- finally, a comparison will be done between the solutions $s$ and $s'$.

Figure B.8, Figure B.9, Figure B.10 and Table B.7 allow to compare in a detailed way the two approaches. They respectively show the improvement brought by the new approach according to the number of tasks, the number of processors, the CCR, and the processor heterogeneity. Experiments show that our approach improves on average the results obtained by ECS. Indeed, as shown in Table B.7, the energy consumption is reduced by **47.49%** and the makespan reduced by of **12.05%**. In addition, Figure B.9 shows clearly that the more processors there are, the more the new approach improves the results of ECS.

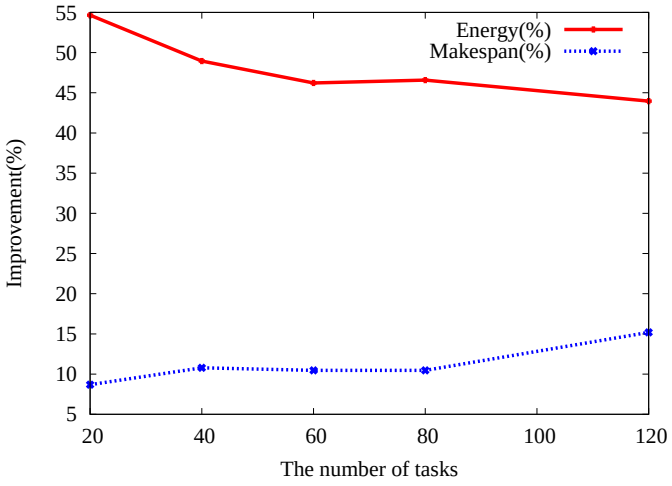



Figure B.8: Improvement according to the number of tasks

Table B.7: Improvement according to the processor heterogeneity

| Processor heterogeneity | Energy (%) | Makespan (%) |
|---|---|---|
| 100 | 44.74 | 9.10 |
| 200 | 43.49 | 7.07 |
| random | 54.26 | 19.99 |
| **Average** | **47.49** | **12.05** |

### B.5.3 Insular approach

The objective of the following experiments is to show that our island approach improves the quality of the solutions provided by the hybrid approach. This insular

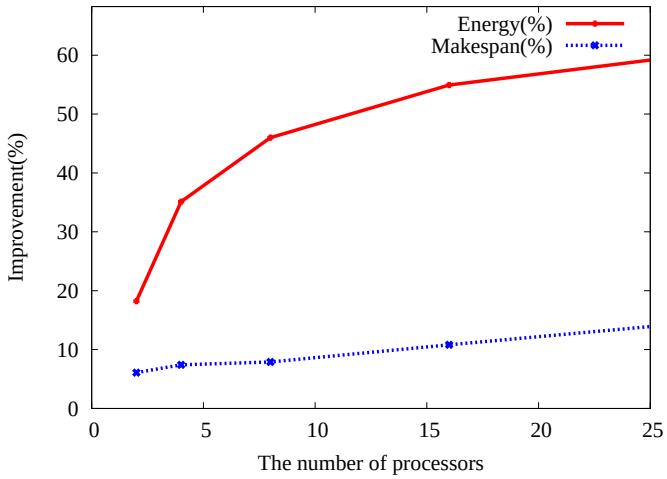



Figure B.9: Improvement according to the number of processors

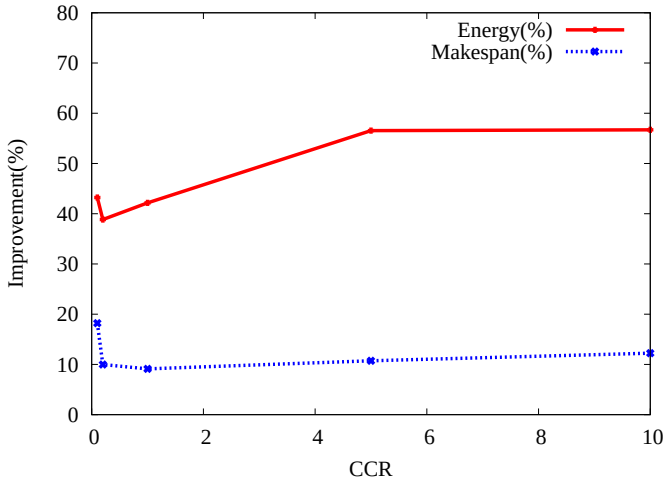



Figure B.10: Improvement according to the CCR

approach is useful when solving large instances. Therefore, the experiments, presented in this section, focus only on the large instances of Table B.4. The instances used are those with the number of tasks is 120, the number of processors is 64, the value of CCR is 10, and the heterogeneity of processors is 200 (20 instances). Each instance is solved using 1, 5, 10, 30 or 50 islands. An insular approach with 1 island is equivalent to the hybrid approach.

Figure B.11 illustrates the S-metric average values obtained with different numbers of islands. These values are normalized with the average value obtained by the experiments using 1 island. The S-metric measures the hyper-volume defined by a reference point and a Pareto front. It allows to evaluate the quality of a Pareto front provided by an algorithm.

Experiments show that whatever the number of used islands the insular approach improves the Pareto front obtained with the hybrid approach. As shown in Figure B.11, the use of 50 islands, instead of 1 island (i.e. the hybrid approach), improves the S-metric of the obtained Pareto front by **26%**. In Figure B.11, the more the number of islands is used, the better the results will be.

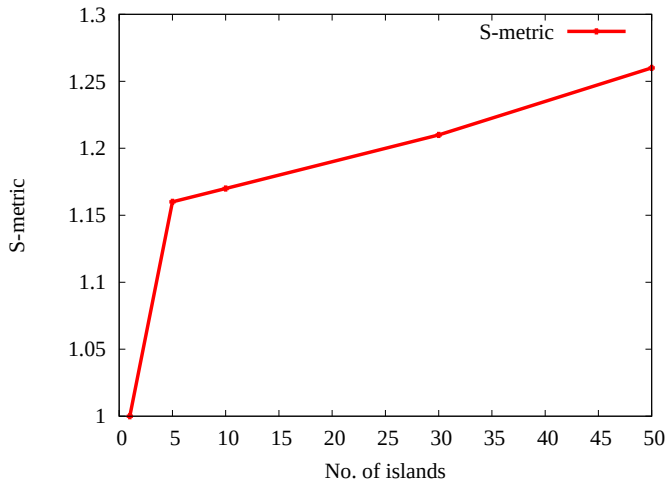



Figure B.11: S-metric value according to the number of islands

### B.5.4 Multi-start approach

This section presents the experiments done to assess the quality of our multi-start approach. The parameters of the instances used in our experiments are: the CCR is 0.1, 0.5, 1.0, 5.0 or 10.0, the number of processors is 8, 32, or 64, and the heterogeneity of processors is 100, 200 or random. The population of the GA contains 20 chromosomes. Therefore, 21 computing cores are used to solve each instance (20 cores to run the ECSs and 1 core to run the GA). In our case, an experiment can not have a speedup greater than 21.

Table B.8, Figure B.12, Figure B.13 show respectively the evolution of the speedup according to the processor heterogeneity, the CCR, and the number of

processors. The average speedup obtained is 13.06.

As shown in Figure B.13, the speedup increases proportionally to the number of processors on which the precedence-constrained parallel application is run. Table B.8 and Figure B.13 that show the CCR and the heterogeneity of processors do not impact significantly the quality of the acceleration of our approach.

Table B.8: Speedup according to the processor heterogeneity

| Processor heterogeneity | Speedup |
|---|---|
| 100 | 13.74 |
| 200 | 12.73 |
| random | 12.73 |
| **Average speedup** | **13.06** |

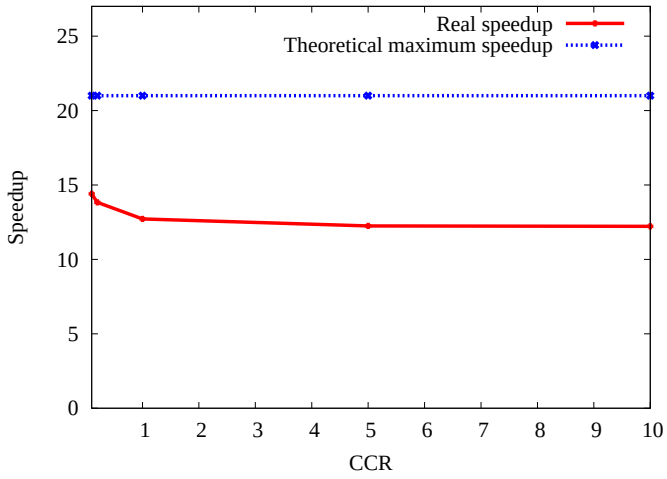



Figure B.12: Speedup according to the CCR

## B.6 Conclusions

In this paper, we have investigated the precedence-constrained parallel applications particularly on high-performance computing systems like cloud computing infrastructures. Precedence-constrained parallel applications are designed mostly with the sole goal of minimizing completion time without paying much attention to energy consumption.

We presented a new parallel bi-objective hybrid genetic algorithm to solve this problem. The algorithm minimizes energy consumption and makespan. The energy saving of our approach exploits the dynamic voltage scaling (DVS) technique a recent advance in processor design.

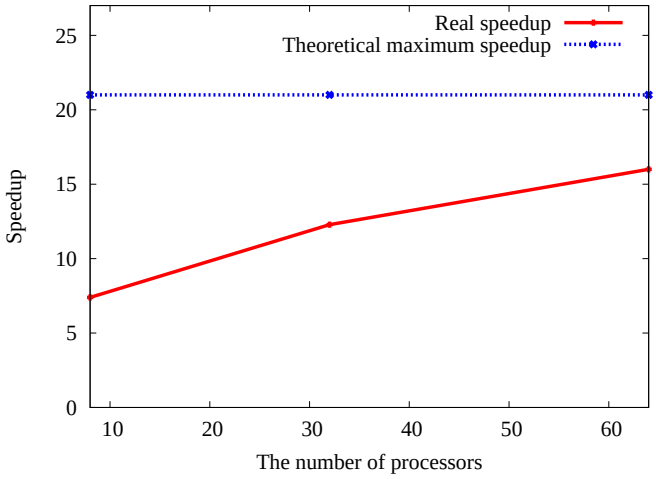



Figure B.13: Speedup according to the number of processors

Our new approach has been evaluated with the Fast Fourier Transformation task graph which is a real-world application. Experiments show that our bi-objective meta-heuristic improves on average the results obtained in the literature (see [Topcuouglu 2002], [Bozdag 2006] and [Garey 1979]) particularly in energy saving. Indeed, the energy consumption is reduced by **47.5%** and the completion time by **12%**. The experiments of the insular approach also show that the more the number of islands is used, the better the results will be. The use of 50 islands, instead of 1 island (i.e. the hybrid approach), improves the S-metric of the obtained Pareto front by **26%**. Furthermore, the multi-start approach is on average **13** times faster than the island approach using 21 cores.

However, we observed that the hybrid approach consumes more resources than ECS, and the insular approach consumes more resources than the hybrid approach. In the insular approach, experiments show that the more the number of islands is used, the more the resources are needed. A resource can be a processor, a network bandwidth, etc. The energy consumed by an approach increases when the used resources increase. We think that the multi-start approach does not increase significantly the energy consumed by the insular approach.

Therefore, one of the main perspectives of the work presented in this paper is to determine the solving approach to choose among ECS, the hybrid approach, and the insular approach, according to the precedence-constrained parallel application at hand. If the insular approach is chosen, the major issue is to determine the best number of islands to be used. This future work aims to minimize the total amount of consumed energy by the chosen solving approach and by the precedence-constrained parallel application to be solved. It is clear, for example, that the insular approach is interesting for the large and resource consuming precedence-constrained parallel applications and the applications intended to be executed several times.

# Appendix

**Algorithm 10** The main parameters of our approaches

```
1: INSULAR←TRUE
2: MULTISTART←TRUE
3:
4: GENERATION_MAXIMUM←200
5: POPULATION_SIZE←20
6: CROSSOVER_RATE←1
7: MUTATION_RATE←0.35
8: N←size(solution)
9:
10: MIGRATION_TOPOLOGY←RING
11: MIGRATION_RATE←20
12: MIGRANTS_SIZE←5
```

**Algorithm 11** The main algorithm of our approaches

**hybrid_insular_multistart_approches()**

```
1: initialize(population,POPULATION_SIZE)
2: evaluate(population)
3: GENERATION←1
4: while GENERATION ≤ GENERATION_MAXIMUM do
5:    parents←select(population)
6:    children←crossover(parents)
7:    mutation(children)
8:    evaluate(children)
9:    replace(population,children)
10:   update(archive,children)
11:   migrate(population,GENERATION)
12:   GENERATION←GENERATION+1
```

**Algorithm 12** Fitness operator

**evaluate(population)**

```
1: if MULTISTART then
2:    evaluate_parallel(population)
3: else
4:    evaluate_sequential(population)
```

**evaluate_parallel(population)**

```
1: for all solution ∈ population do
2:    launch_parallel(ECS_component2,solution)
3: for all solution ∈ population do
4:    cost(solution)←read_cost(solution)
```

**evaluate_sequential(population)**

```
1: for all solution ∈ population do
2:    cost(solution)←ECS_component2(solution)
```

**Algorithm 13** Mutation operator

**mutation(children)**

```
1: for all solution ∈ children do
2:    if random([0,1]) ≤ MUTATION_RATE then
3:       mutation(solution)
```

**mutation(solution)**

```
1: (i,j)=random(1≤i<j≤N ∧ check_level(solution,i,j))
2: swap(task(solution_i),task(solution_j))
```

**check_level(solution,i,j)**

```
1: OUTPUT: b-level(task(solution_i))<b-level(task(solution_j))
```

**Algorithm 14** Crossover operator

**crossover(parents)**

```
1: children← ∅
2: for all i ∈ (1...POPULATION_SIZE) do
3:    if random([0,1]) ≤ CROSSOVER_RATE then
4:       (parent1,parent2)←select(parents)
5:       (child1,child2)←crossover2(parent1,parent2)
6:       add(children,child1)
7:       add(children,child2)
8: OUTPUT: children
```

**crossover2(parent1,parent2)**

```
1: child1←crossover1(parent1,parent2)
2: child2←crossover1(parent2,parent1)
3: OUTPUT: (child1,child2)
```

**crossover1(parent1,parent2)**

```
1: (i,j)=random(1≤i<j≤N)
2: for all k ∈ (1,...,i-1,j+1,...,N) do
3:    task(child_k)←task(parent1_k)
4: m←0
5: for all [k ∈ (1,..,N)] ∧ [task(parent2_k) ∉ tasks(child)] do
6:    task(solution_buffer_m)←task(parent2_k)
7:    m←m+1
8: for all k ∈ (i,...,j) do
9:    task(child_k)←task(solution_buffer_{k−i+1})
10: OUTPUT: child
```

**Algorithm 15** Migration operator

**migrate(population,GENERATION)**

```
1: if NOT INSULAR then
2:    stop
3: if GENERATION mod(MIGRATION_RATE)≠0 then
4:    stop
5: migrants←select(population,MIGRANTS_SIZE)
6: DESTINATION←destination(topology)
7: send(DESTINATION,migrants)
8: SOURCE←source(topology)
9: migrants←receive(SOURCE)
10: insert(population,migrants)
```

# Bibliography


[Ama 2012a] *Amazon EC2 Pricing*. http://aws.amazon.com/fr/ec2/pricing/, 2012. (Cited on pages 3 and 75.)

[Ama 2012b] *Amazon Elastic Compute Cloud (Amazon EC2)*. http://aws.amazon.com/fr/ec2/, 2012. (Cited on pages 2, 11, 36, 39 and 100.)

[Ama 2013] *Amazon Elastic Compute Cloud (Amazon EC2) Instance type*. http://aws.amazon.com/en/ec2/instance-types/, 2013. (Cited on pages 65, 74 and 100.)

[AMD 2004] AMD. *Cool'n'Quiet Real world numbers*. http://icrontic.com/article/socket_940_vs_939, 2004. (Cited on page 88.)

[Andreolini 2010] M. Andreolini, S. Casolari, M. Colajanni and M. Messori. *Dynamic Load Management of Virtual Machines in Cloud Architectures*. In R.D. Avresky, M. Diaz, A. Bode, B. Ciciani and E. Dekel, editeurs, Cloud Computing, volume 34 of *LNICST*, pages 201–214. Springer Berlin Heidelberg, 2010. (Cited on pages 29 and 31.)

[Armbrust 2009] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin and M. Zaharia. *Above the Clouds: A Berkeley View of Cloud Computing*. Rapport technique, 2009. (Cited on pages 1 and 7.)

[Barham 2003] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt and A. Warfield. *Xen and the art of virtualization*. In SOSP, pages 164–177, 2003. (Cited on page 8.)

[Borgetto 2012a] D. Borgetto, H. Casanova, G. Da Costa and J.M Pierson. *Energy-aware service allocation*. Future Gener. Comput. Syst., vol. 28, no. 5, pages 769–779, May 2012. (Cited on page 30.)

[Borgetto 2012b] D. Borgetto, M. Maurer, G. Da-Costa, J.M Pierson and I. Brandic. *Energy-efficient and SLA-aware management of IaaS clouds*. In Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet, e-Energy '12, pages 25:1–25:10, New York, NY, USA, 2012. ACM. (Cited on page 29.)

[Bozdag 2006] D. Bozdag, U. Catalyurek and F. Ozguner. *A task duplication based bottom-up scheduling algorithm for heterogeneous environments*. In Parallel and Distributed Processing 20th International Symposium (IPDPS), April 2006. (Cited on pages 132, 137 and 151.)

[Bunde 2006] D.P. Bunde. *Power-aware scheduling for makespan and flow.* In Proc. the eighteenth annual ACM symposium on Parallelism in algorithms and architectures, July 2006. (Cited on pages 131 and 132.)

[Burd 1995] T.D. Burd and R.W. Brodersen. *Energy efficient CMOS microprocessor design.* In System Sciences. Proceedings of the 28th Hawaii International Conference on, pages 288–297 vol.1, 1995. (Cited on pages 37 and 88.)

[Burge 2007] J. Burge, P. Ranganathan and J.L. Wiener. *Cost-aware scheduling for heterogeneous enterprise machines (CASH EM).* In Cluster Computing, IEEE International Conference on, pages 481–487, 2007. (Cited on pages 14, 25 and 31.)

[Buyya 2008] R. Buyya, Chee Shin Yeo and S. Venugopal. *Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities.* In High Performance Computing and Communications (HPCC) 10th IEEE International Conference on, pages 5–13, 2008. (Cited on pages 1 and 8.)

[Buyya 2009] R. Buyya, S. Pandey and C. Vecchiola. *Cloudbus Toolkit for Market-Oriented Cloud Computing.* In Martin.Gilje Jaatun, Gansen Zhao and Chunming Rong, editeurs, Cloud Computing, volume 5931 of *Lecture Notes in Computer Science*, pages 24–44. Springer Berlin Heidelberg, 2009. (Cited on page 9.)

[Cahon 2004] S. Cahon, N. Melab and E-G. Talbi. *ParadisEO: A Framework for the Reusable Design of Parallel and Distributed meta-heuristics.* Journal of Heuristics, vol. 10, no. 3, pages 357–380, May 2004. (Cited on pages 19, 21 and 144.)

[Cameron 2010] K.W. Cameron. *Trading in green IT.* Computer, vol. 43, no. 3, pages 83–85, March 2010. (Cited on page 132.)

[Campbell 2009] R. Campbell, I. Gupta, M. Heath, S.Y. Ko, M. Kozuch, M. Kunze, T. Kwan, K. Lai, H.Y. Lee, M. Lyons, D. Milojicic, D. O'Hallaron and Y.C. Soh. *Open Cirrus cloud computing testbed: federated data centers for open source systems and services research.* In Proceedings of the conference on Hot topics in cloud computing (HotCloud), Berkeley, CA, USA, 2009. USENIX Association. (Cited on pages 1 and 35.)

[Chaisiri 2009] S. Chaisiri, Bu-Sung Lee and D. Niyato. *Optimal virtual machine placement across multiple cloud providers.* In Services Computing Conference, IEEE Asia-Pacific (APSCC), pages 103–110, dec. 2009. (Cited on page 26.)

[Chen 2005a] J. J. Chen and T. W. Kuo. *Multiprocessor Energy-Efficient Scheduling for Real-Time Tasks with Different Power Characteristics.* In International

Conference on Parallel Processing (ICPP), pages 13–20, 2005. (Cited on page 137.)

[Chen 2005b] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang and N. Gautam. *Managing server energy and operational costs in hosting centers*. SIGMETRICS Perform. Eval. Rev., vol. 33, no. 1, pages 303–314, June 2005. (Cited on pages 38 and 88.)

[Chen 2011] J. Chen, C. Wang, B.B. Zhou, L. Sun, Y.C. Lee and A.Y. Zomaya. *Tradeoffs Between Profit and Customer Satisfaction for Service Provisioning in the Cloud*. In HPDC, ACM, pages 229–238, New York, NY, USA, 2011. (Cited on pages 27, 31, 66 and 75.)

[Chun 2002] B.N. Chun and D.E. Culler. *User-Centric Performance Analysis of Market-Based Cluster Batch Schedulers*. In Cluster Computing and the Grid, 2nd IEEE/ACM International Symposium on, page 30, may 2002. (Cited on page 25.)

[Cohoon 1987] J.P. Cohoon, S.U. Hedge, W.N. Martin and D. Richards. *Punctuated equilibria : A parallel genetic algorithm*. In J.J. Grefenstette and Lawrence Erlbaum Associates, editeurs, Proceedings of the Second International Conference on Genetic Algorithms, page 148, 1987. (Cited on page 142.)

[Cormen 1990] T.H. Cormen, C.E. Leiserson and R.L. Rivest. *Introduction to Algorithms*. In MIT Press, 1990. (Cited on page 143.)

[Crainic 2003] T.G. Crainic and M. Toulouse. *Parallel Strategies for Meta-Heuristics*. In Fred Glover and Gary.A. Kochenberger, editeurs, Handbook of Metaheuristics, volume 57 of *International Series in Operations Research & Management Science*, pages 475–513. Springer US, 2003. (Cited on page 17.)

[Darbha 1998] S. Darbha and D. P. Agrawal. *Optimal scheduling algorithm for distributed-memory machines*. IEEE Trans. Parallel Dist. Systems, vol. 9, no. 1, pages 87–95, 1998. (Cited on page 131.)

[Deb 2001] K. Deb. Multi-objective optimization using evolutionary algorithms. Wiley, 2001. (Cited on page 139.)

[Deb 2002] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan. *A fast and elitist multiobjective genetic algorithm: NSGA-II*. Evolutionary Computation, IEEE Transactions on, vol. 6, no. 2, pages 182–197, 2002. (Cited on pages 45 and 71.)

[DOE 2007] *US Department of Energy, Voluntary reporting of greenhouse gases: Appendix F.* http://www.eia.doe.gov/oiaf/1605/pdf/Appendix20F_r071023.pdf, 2007. (Cited on page 48.)

[Edgeworth 1881] F. Y. Edgeworth. *C. Kegan Paul and Co: An Essay on the Application of Mathematics to the Moral Sciences*. Mathematical Psychics, London, 1881. (Cited on page 14.)

[Eff 2011] *Efficap energie*. http://www.efficap-energie.com, 2011. (Cited on page 34.)

[EIA 2007] *US Energy Information Administration (EIA) report*. http://www.eia.doe.gov/cneaf/electricity/epm/table5_6_a.html, 2007. (Cited on page 48.)

[Elmroth 2009] E. Elmroth, F.G. Marquez, D. Henriksson and D.P. Ferrera. *Accounting and Billing for Federated Cloud Infrastructures*. In Grid and Cooperative Computing (GCC), 8th International Conference on, pages 268 –275, aug. 2009. (Cited on pages 26 and 31.)

[Feitelson 2009] D. Feitelson. *Parallel workloads archive*. http://www.cs.huji.ac.il/labs/parallel/workload, Aug 2009. (Cited on pages 3 and 48.)

[Feller 2012] E. Feller, L. Rilling and C. Morin. *Snooze: A Scalable and Autonomic Virtual Machine Management Framework for Private Clouds*. In Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pages 482–489, Washington, DC, USA, 2012. (Cited on pages 14, 30 and 31.)

[Foster 2008] I. Foster, Yong Zhao, I. Raicu and Shiyong Lu. *Cloud Computing and Grid Computing 360-Degree Compared*. In Grid Computing Environments Workshop., pages 1–10, 2008. (Cited on page 5.)

[Garey 1979] M.R. Garey and D.S. Johnson. Computers and intractability: A guide to the theory of np-completeness. W. H. Freeman & Co., New York, NY, USA, 1979. (Cited on pages 6, 38, 67, 91, 137 and 151.)

[Garg 2008] S. Garg, P. Konugurthi and R. Buyya. *A Linear Programming Driven Genetic Algorithm for Meta-Scheduling on Utility Grids*. In Advanced Computing and Communications. ADCOM. 16th International Conference on, pages 19–26, 2008. (Cited on pages 14, 25 and 31.)

[Garg 2010] S.K. Garg, C.S. Yeo, A. Anandasivam and R. Buyya. *Environment-conscious scheduling of HPC applications on distributed Cloud-oriented data centers*. Journal of Parallel and Distributed Computing, vol. In Press, Corrected Proof, pages –, 2010. (Cited on pages 1, 16, 26, 31 and 48.)

[Gartner 2007] Gartner. *Gartner Estimates ICT Industry Accounts for 2 Percent of Global CO2 Emissions*. http://www.gartner.com/it/page.jsp?id=503867, 2007. (Cited on page 34.)

[Ge 2005] R. Ge, X. Feng and K.W. Cameron. *Performance-constrained Distributed DVS Scheduling for Scientific Applications on Power-aware Clusters*. In Proc. the ACM/IEEE Conference on Supercomputing, pages 34–44, November 2005. (Cited on pages 131, 132 and 137.)

[GoG 2013] *GoGrid Cloud Hosting*. http://www.gogrid.com/, 2013. (Cited on page 11.)

[Greenberg 2006] S. Greenberg, E. Mills, B. Tschudi, P. Rumsey and B. Myatt. *Best practices for data centers: results from benchmarking 22 data centers*. In Proceedings of the ACEEE Summer Study on Energy Efficiency in Buildings, Pacific Grove, USA, 2006. (Cited on page 48.)

[GRI 2013] *GRID5000*. https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home, 2013. (Cited on pages 3, 100 and 102.)

[Guzek 2012] M. Guzek, C. Diaz, J. Pecero, P. Bouvry and A.Y. Zomaya. *Impact of Voltage Levels Number for Energy-Aware Bi-objective DAG Scheduling for Multi-processors Systems*. In Borworn Papasratorn, Nipon Charoenkitkarn, Kittichai Lavangnananda, Wichian Chutimaskul and Vajirasak Vanijja, editeurs, Advances in Information Technology, volume 344 of *Communications in Computer and Information Science*, pages 70–80. Springer Berlin Heidelberg, 2012. (Cited on page 28.)

[Hamilton 2009] J. Hamilton. *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, 2009. (Cited on page 34.)

[Humeau 2013] Jérémie Humeau, Arnaud Liefooghe, El-Ghazali Talbi and Sébastien Verel. *ParadisEO-MO: From Fitness Landscape Analysis to Efficient Local Search Algorithms*. Journal of Heuristics, page (to appear), 2013. (Cited on page 20.)

[Intel 2004] Intel. *Intel Pentium M Processor datasheet*, 2004. (Cited on page 133.)

[Irwin 2004] D.E. Irwin, L.E. Grit and J.S. Chase. *Balancing risk and reward in a market-based task service*. In High performance Distributed Computing proceedings, 13th IEEE International Symposium on, pages 160–169, june 2004. (Cited on page 25.)

[Iverson 1999] M.A. Iverson, F. Özgüner and L.C. Potter. *Statistical Prediction of Task Execution Times Through Analytic Benchmarking for Scheduling in a Heterogeneous Environment*. IEEE Transactions on Computers, vol. 48, pages 1374–1379, 1999. (Cited on page 36.)

[Johnson 1973] D.S. Johnson. *Near-optimal bin packing algorithms*. PhD thesis, Massachusetts Institute of Technology, 1973. (Cited on page 50.)

[Keahey 2007] K. Keahey, T. Freeman, J. Lauret and D. Olson. *Virtual workspaces for scientific applications*. J. Phys.: Conf. Ser., vol. 78, no. 1, pages –, 2007. (Cited on page 11.)

[Keijzer 2002] M. Keijzer, J.J. Merelo, G. Romero and Marc Schoenauer. *Evolving Objects: A General Purpose Evolutionary Computation Library*. In Pierre Collet, Cyril Fonlupt, Jin-Kao Hao, Evelyne Lutton and Marc Schoenauer, editeurs, Artificial Evolution, volume 2310 of *Lecture Notes in Computer Science*, pages 231–242. Springer Berlin Heidelberg, 2002. (Cited on page 19.)

[Khan 2009] S.U. Khan and I. Ahmad. *A Cooperative Game Theoretical Technique for Joint Optimization of Energy Consumption and Response Time in Computational Grids*. IEEE Transactions on Parallel and Distributed Systems, vol. 20, no. 3, pages 346–360, 2009. (Cited on pages 16 and 138.)

[Kim 2007a] K. H. Kim, R. Buyya and J. Kim. *Power Aware Scheduling of Bag-of-Tasks Applications with Deadline Constraints on DVS-enabled Clusters*. In Proc. the Seventh IEEE Int Symp. Cluster Computing and the Grid, May 2007. (Cited on pages 131, 137 and 138.)

[Kim 2007b] S. C. Kim, S. Lee and J. Hahm. *Push-Pull: Deterministic Search-Based DAG Scheduling for Heterogeneous Cluster Systems*. IEEE Trans. Parallel Dist. Systems, vol. 18, no. 11, pages 1489–1502, 2007. (Cited on page 139.)

[Kliazovich 2012] D. Kliazovich, P. Bouvry and S.U. Khan. *Simulating communication processes in energy-efficient cloud computing systems*. In Cloud Networking (CLOUDNET), pages 215–217, 2012. (Cited on page 37.)

[Koch 2005] G. Koch. *Discovering multi-core: Extending the benefits of Moore's law*. In Technology@Intel Magazine, July 2005. (Cited on page 132.)

[Koomey 2008] J. G. Koomey. *Estimating total power consumption by servers in the U.S. and the world*, 2008. (Cited on pages 34, 86 and 132.)

[Kwok 1998] Y. K. Kwok and I. Ahmad. *Benchmarking the Task Graph Scheduling Algorithms*. In Proc. First Merged Int Parallel Symposium on Parallel and Distributed Processing (IPPS/SPDP), pages 531–537, 1998. (Cited on page 137.)

[Laszewski 2009] G. Von Laszewski, Lizhe Wang, Andrew J. Younge and Xi He. *Power-Aware Scheduling of Virtual Machines in DVFS-enabled Clusters*. In Cluster Computing and Workshops (CLUSTER), IEEE International Conference on, pages 1–10, New Orleans, LA, 2009. (Cited on pages 2, 29 and 31.)

[Lee 2005] Y.C. Lee and A.Y. Zomaya. *A Productive Duplication-Based Scheduling Algorithm for Heterogeneous Computing Systems*. In Proceedings of the First international conference on High Performance Computing and Communications (HPCC), pages 203–212, 2005. (Cited on page 139.)

[Lee 2008] Y.C. Lee and A.Y. Zomaya. *A Novel State Transition Method for metaheuristic-Based Scheduling in Heterogeneous Computing Systems.* vol. 19, no. 9, pages 1215–1223, September 2008. (Cited on page 131.)

[Lee 2009] Y.C Lee and A.Y Zomaya. *Minimizing Energy Consumption for Precedence-Constrained Applications Using Dynamic Voltage Scaling.* In Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID), pages 92–99, Washington, DC, USA, 2009. IEEE Computer Society. (Cited on pages 15, 28, 132, 138 and 140.)

[Lee 2010a] Y.C. Lee, C. Wang, A.Y. Zomaya and B.B. Zhou. *Profit-Driven Service Request Scheduling in Clouds.* In Cluster, Cloud and Grid Computing (CCGRID), 10th IEEE/ACM International Conference on, pages 15 –24, May 2010. (Cited on pages 25 and 31.)

[Lee 2010b] Y.C Lee and A.Y Zomaya. *Energy efficient utilization of resources in cloud computing systems.* The Journal of Supercomputing, pages 1–13, 2010. 10.1007/s11227-010-0421-3. (Cited on pages 28 and 31.)

[Liefooghe 2009] Arnaud Liefooghe, Laetitia Jourdan and El-Ghazali Talbi. *A Unified Model for Evolutionary Multiobjective Optimization and its Implementation in a General Purpose Software Framework: ParadisEO-MOEO.* Rapport de recherche RR-6906, INRIA, 2009. (Cited on page 20.)

[Lucas-Simarro 2012] J. Luis Lucas-Simarro, R. Moreno-Vozmediano, R.S. Montero and I.M. Llorente. *Scheduling strategies for optimal service deployment across multiple clouds.* Future Generation Computer Systems, no. 0, pages –, 2012. (Cited on pages 26 and 31.)

[Mankiw 2008] G. Mankiw. Principles of economics. Sourth-Western Pub, 2008. (Cited on pages 66 and 83.)

[Mezmaz 2011] M. Mezmaz, N. Melab, Y. Kessaci, Y.C. Lee, E.-G. Talbi, A.Y. Zomaya and D. Tuyttens. *A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems.* Journal of Parallel and Distributed Computing, vol. In Press, Corrected Proof, pages –, 2011. (Cited on page 28.)

[Mills 2011] K. Mills, J. Filliben and C. Dabrowski. *Comparing VM-Placement Algorithms for On-Demand Clouds.* In Proceedings of the IEEE Third International Conference on Cloud Computing Technology and Science (CLOUDCOM), pages 91–98, Washington, DC, USA, 2011. IEEE Computer Society. (Cited on pages 29 and 94.)

[Milojicic 2011] D. Milojicic, I.M. Llorente and R.S. Montero. *OpenNebula: A Cloud Management Tool.* Internet Computing, IEEE, vol. 15, no. 2, pages 11–14, 2011. (Cited on page 11.)

[Min 2000] R. Min, T. Furrer and A. Chandrakasan. *Dynamic Voltage Scaling Techniques for Distributed Microsensor Networks*. In Proc. IEEE Workshop on VLSI, pages 43–46, April 2000. (Cited on page 133.)

[Nurmi 2009] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff and D. Zagorodnov. *The Eucalyptus Open-Source Cloud-Computing System*. In Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID), pages 124–131, Washington, DC, USA, 2009. IEEE Computer Society. (Cited on page 11.)

[Ope 2012] *Openstack: open source cloud computing software*. http://www.openstack.org/, 2012. (Cited on page 11.)

[Orgerie 2008] A-C. Orgerie, L. Lefevre and J.-P. Gelas. *Save Watts in Your Grid: Green Strategies for Energy-Aware Framework in Large Scale Distributed Systems*. In 14th IEEE International Conference on Parallel and Distributed Systems (ICPADS), pages 171–178, 2008. (Cited on page 30.)

[Pareto 1896] V. Pareto. *Cours d'économie politique*. Rouge, Lausanne, Switzerland., vol. 1-2, 1896. (Cited on pages 12 and 14.)

[Pillai 2001] P. Pillai and 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), pages 89–102, New York, NY, USA, 2001. (Cited on pages 37 and 88.)

[Rajpathak 2001] D.G. Rajpathak. *Knowledge Media Institute Intelligent scheduling - A Literature Review*. 2001. (Cited on page 21.)

[Rimal 2009] B.P. Rimal, Eunmi Choi and I. Lumb. *A Taxonomy and Survey of Cloud Computing Systems*. In INC, IMS and IDC. NCM. 5th International Joint Conference on, pages 44–51, 2009. (Cited on pages 9 and 12.)

[Rizvandi 2010] N.B. Rizvandi, J. Taheri, A.Y. Zomaya and 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, vol. 0, pages 388–397, 2010. (Cited on pages 16, 28 and 31.)

[Rountree 2007] B. Rountree, D. K. Lowenthal, S. Funk, V. W. Freeh, B. R. de Supinski and M. Schulz. *Bounding energy consumption in large-scale MPI programs*. In Proc. the ACM/IEEE conference on Supercomputing, November 2007. (Cited on pages 131, 137 and 138.)

[Srikantaiah 2008] S. Srikantaiah, A. Kansal and F. Zhao. *Energy Aware Consolidation for Cloud Computing*. In Proceedings of Workshop on Power Aware Computing and Systems (HotPower). USENIX, December 2008. (Cited on page 28.)

[Talbi 2009] E-G. Talbi. Metaheuristics : from design to implementation. The Sciences Po series in international relations and political economy. John Wiley & Sons, 2009. (Cited on pages 16, 19, 45, 71 and 144.)

[Tesauro 2007] G. Tesauro, R. Das, H. Chan, J.O. Kephart, D. Levine, F.L. Rawson III and C. Lefurgy. *Managing Power Consumption and Performance of Computing Systems Using Reinforcement Learning.* In NIPS, 2007. (Cited on page 28.)

[Topcuouglu 2002] H. Topcuouglu, S. Hariri and M.Y. Wu. *Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing.* IEEE Trans. Parallel Dist. Systems, vol. 13, no. 3, pages 260–274, 2002. (Cited on pages 131, 132, 137 and 151.)

[Tordsson 2012] J. Tordsson, R.S. Montero, R. Moreno-Vozmediano and I.M. Llorente. *Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers.* Future Generation Computer Systems, vol. 28, no. 2, pages 358–367, 2012. (Cited on pages 26 and 31.)

[Venkatachalam 2005] V. Venkatachalam and M. Franz. *Power reduction techniques for microprocessor systems.* ACM Computing Survey, vol. 37, no. 3, pages 195–237, September 2005. (Cited on page 132.)

[Venugopal 2008] S. Venugopal, Xingchen Chu and R. Buyya. *A Negotiation Mechanism for Advance Resource Reservations Using the Alternate Offers Protocol.* In Quality of Service (IWQoS), 16th International Workshop on, pages 40 –49, June 2008. (Cited on page 48.)

[Verma 2008] A. Verma, P. Ahuja and A. Neogi. *pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems.* In Valérie Issarny and Richard Schantz, editeurs, Middleware 2008, volume 5346 of *Lecture Notes in Computer Science*, pages 243–264. Springer Berlin Heidelberg, 2008. (Cited on pages 2, 29, 31, 91 and 94.)

[Vir 2013] *VMWare: Migrate Virtual machines with Zero Downtime.* http://www.vmware.com/, 2013. (Cited on page 8.)

[Wang 2008] L. Wang and Y. Lu. *Efficient Power Management of Heterogeneous Soft Real-Time Clusters.* In Real-Time Systems Symposium, pages 323–332, 2008. (Cited on pages 38 and 88.)

[Wang 2010] L. Wang, G. von Laszewski and J. Dayal. *Towards Energy Aware Scheduling for Precedence Constrained Parallel Tasks in a Cluster with DVFS.* pages 17–20, May 2010. (Cited on page 138.)

[Wu 2013] Z. Wu, X. Liu, Z. Ni, D. Yuan and Y. Yang. *A market-oriented hierarchical scheduling strategy in cloud workflow systems.* The Journal of

Supercomputing, vol. 63, no. 1, pages 256–293, 2013. (Cited on pages 1, 2, 27 and 31.)

[Xu 2010] J. Xu and J.A.B Fortes. *Multi-Objective Virtual Machine Placement in Virtualized Data Center Environments*. In IEEE/ACM Int Conference Green Computing and Communications (GreenCom) & Int Conference on Cyber, Physical and Social Computing (CPSCom), pages 179–188, 2010. (Cited on pages 2, 16, 30 and 31.)

[Yu 2006] J. Yu and R. Buyya. *Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms*. Scientific Programming, vol. 14, no. 3-4, pages 217–230, 2006. (Cited on pages 25 and 31.)

[Zhong 2007] X. Zhong and C.-Z. Xu. *Energy-aware modeling and scheduling for dynamic voltage scaling with statistical real-time guarantee*. IEEE Trans. Computers, vol. 56, no. 3, pages 358–372, 2007. (Cited on page 137.)

[Zhu 2003] D. Zhu, R. Melhem, and B. R. Childers. *Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems*. IEEE Trans. Parallel Dist. Systems, vol. 14, no. 7, pages 686–700, 2003. (Cited on pages 131, 132 and 137.)

[Zhu 2004] D. Zhu, D. Mosse and R. Melhem. *Power-aware scheduling for AND/OR graphs in real-time systems*. IEEE Trans. Parallel Dist. Systems, vol. 15, no. 9, pages 849–864, 2004. (Cited on pages 131 and 137.)

[Zhuo 2008] J. Zhuo and C. Chakrabarti. *Energy-efficient dynamic task scheduling algorithms for DVS systems*. ACM Trans. Embed. Comput. Syst., vol. 7, pages 17:1–17:25, January 2008. (Cited on page 14.)

[Zomaya 1999] A.Y. Zomaya, C. Ward, and B.S. Macey. *Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues*. IEEE Trans. Parallel Dist. Systems, vol. 10, no. 8, pages 795–812, 1999. (Cited on page 131.)

**Abstract:** Cloud computing has emerged during the last decade to be widely adopted nowadays in several IT areas. It consists to propose market or not market-oriented resources as services that can be consumed in a ubiquitous, flexible and transparent way. In this PhD thesis, we deal with scheduling, one of the major cloud computing issue. According to the targeted cloud configuration, we have identified three levels of scheduling: service-level, task-level and Virtual Machine-level. We revisit the problem modeling, the design and the implementation of multi-objective metaheuristics for each scheduling level of the cloud. The proposed metaheuristics-based schedulers address different criteria including energy consumption, greenhouse gas emissions, profit and QoS (cost and response time). We prove their adaptability to the cloud constraints by integrating them as a part of the OpenNebula cloud manager. Moreover, our schedulers have been extensively experimented using realistic cloud configurations on Grid'5000, considered as an infrastructure as a service (IAAS), and concrete scenarios based on Amazon EC2 instances and prices. The reported results show that our proposed methods outperform existing scheduling approaches in terms of all previously cited criteria.
**Keywords:** resource scheduling, cloud computing, evolutionary algorithms , local search, multi-objective optimization, metaheuristics

---

**Résumé:** Le cloud computing a émergé au cours de la dernière décennie pour être largement adopté aujourd'hui dans plusieurs domaines de l'informatique. Il consiste à proposer des ressources axées, ou non, sur le marché sous forme de services qui peuvent être consommés de manière souple et transparente. Dans cette thèse, nous traitons le problème d'ordonnancement, un des enjeux majeurs du cloud. Selon la configuration de cloud ciblée, nous avons identifié trois niveaux d'ordonnancement : niveau service, niveau tâche et niveau machine virtuelle. Nous revisitons la modélisation du problème, la conception et l'implémentation des métaheuristiques multi-objectives pour chaque niveau d'ordonnancement du cloud. Les ordonnanceurs à base de métaheuristiques que nous proposons portent sur différents critères notamment la consommation d'énergie, les émissions de gaz à effet de serre, le profit et la qualité du service (coût et temps de réponse). Nous prouvons leur capacité d'adaptation aux contraintes du cloud en les intégrant au sein du gestionnaire de cloud OpenNebula. De plus, nos ordonnanceurs ont été largement expérimentés utilisant des configurations réalistes de cloud sur Grid'5000, en tant qu'infrastructure en tant que service (IAAS), et des scénarios concrets basés sur les instances et les tarifications d'Amazon EC2. Les résultats présentés montrent que les méthodes que nous proposons surpassent les approches d'ordonnancement existantes sur tous les critères cités précédemment.
**Mots clés:** ordonnancement de ressources, cloud computing, algorithmes évolutionnaires, recherche locale, optimisation multi-objectif, métaheuristiques