République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université A.MIRA-BEJAIA

Faculté des Sciences Exactes
Département d'Informatique

THÈSE

Présentée par

Abdelhamid Khiat

Pour l'obtention du grade de

DOCTEUR EN SCIENCES

Filière : Informatique

Option : Réseaux et Systèmes Distribués

Thème

Resource Management and Scheduling in Large Scale Distributed Systems: Cloud & Grid

Soutenue le : 04 Février 2021 Devant le Jury composé de :

Nom et Prénom	Grade		
Mr Ahror BELAID	Professeur	Univ. de Béjaia	Président
Mr Abelkamal TARI	Professeur	Univ. de Béjaia	Rapporteur
Mr Nadjib BADACHE	Professeur	USTHB - Alger	Examinateur
Mr Abderrazak SEBAA	MCA	Univ. de Béjaia	Examinateur
Mr Ali MELIT	Professeur	Univ. de Jijel	Examinateur
Mr Azze-Eddine MAREDJ	MRA	CERIST	Examinateur

Année Universitaire : 2020/2021

My parents, wife, and children

ACKNOWLEDGEMENTS

First and foremost, I would like to thank Allah almighty for giving me the strength, knowledge, ability and opportunity to undertake this research study.

I would like to express my deeply-felt thanks to my thesis advisor, Prof. Abdelkamel TARI, for his warm encouragement and thoughtful guidance. It has been an honor and pleasure working with him during the last years.

I also thank the members of thesis committee: Prof. BELAID Ahror, Prof. TARI Abdelkamel, Prof. BADACHE Nadjib, Dr. SEBAA Abderazak, Dr. MELIT Ali and Dr. MAREDJ Azze-eddine for having accepted to assess my thesis.

I am happy to acknowledge my deepest sincere gratitude to Dr. Guérout Tom with whom I exchanged valuable ideas during my visit to LAAS-CNRS and helped me to move forward efficiently in my work preparation.

An extra special recognition to my family whose love and aid have made this thesis possible, especially my parents and my wife, I love you all very much and am immensely grateful for all that you do. Thank you for everything.

I would like to greatly thank again the ex-director of CERIST Prof. BADACHE Nadjib, director of CERIST Prof. Hacène BELBACHIR, and the chief of DRSD division Mrs. EL-MAOUHAB Aouaouche for their support and unbounded trust on me and also for the means they managed to secure for me to fully accomplish my thesis preparation in an efficient way. Without forgetting to acknowledge my dearest colleagues, SADALLAH Madjid, BELAZZOUGUI Djamal, AMRANE Abdesalam, and CHAA Messaoud for the whole time we spent together working, debating, brainstorming, sharing ideas and more.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

LIST OF ALGORITHMS

Algorithms

LIST OF ABBREVIATIONS

GA Genetic Algorithm

ACO Ant Colony Optimization

PSO Particle Swarm Optimization

QoS Quality of Service

HPC High-Performance Computing

InterRC Inter-Resources Collaboration Heuristic

SLA Service Level Agreement

SSH Secure Shell

DEC Digital Equipment Corporation

SNA Systems Network Architecture

NP-Hard Non-deterministic Polynomial-time Hardness

LAAS Laboratoire d'Analyse et d'Architecture des Systèmes

CNRS Centre National de la Recherche Scientifique

PC Personal Computer

RC Relative Cost

RR Round Robin

HCSP Heterogeneous Computing Scheduling Problem

VM Virtual Machine

DAG Directed Acyclic Graph

VO Virtual Organization

SaaS Software as a Service

IaaS Infrastructure as a Service

PaaS Platform as a Service

DaaS Data as a Service

FaaS Function as a Service

BaaS Backend As A Service

IoT Internet of Things

P2P peer-to-peer

CPU Central Processing Unit

GUI Graphical User Interface

MIPS Million Instructions Per Second

MI Millions Instructions

General introduction

Distributed architectures and scheduling

Since the advent of automated computing, computers started to be used in order to realize computing operations and their needs in terms of resources never stopped growing. The arrival of the distributed system was considered as a good solution that can be exploited to realize more complex computing operations, by distributing the computation over a set of computation units that are connected by a network.

First solutions of distributed computing appeared during the 1970s, where many efforts are done to improve this kind of computing system. For example, French Cyclades network, pushed by a French international company called CII and its Distributed System Architecture, based on the Datagram, tried to pool the computing resources of university centers. Meanwhile, In the United States, *IBM* and Digital Equipment Corporation (DEC) created the Systems Network Architecture (SNA) and DECnet architectures, taking advantage of the digitization of AT&T's telephone network and its dedicated medium-speed connections. From that time on, the design of large systems was challenged by networked minicomputers, such as the Mitra and then the Mini, which complemented it and modified its architecture.

Simple computer machines were progressively replaced by distributed computers to the point most of today's computing architectures are distributed. Subsequently, these architectures never stopped evolving, and a huge variety in terms of computing architectures and services offered were created, giving birth to many computing paradigms and architectures, including Cloud, Fog, and a flurry of other heterogeneous distributed computing concepts. The facilities proposed through the services hosted in these kinds of infrastructures are more and more varied, thereby attracting an ever increasing number of users. Each one of these architectures has its own characteristics, use areas, and advantages/disadvantages. This gives the user/provider the freedom to choose which architecture to use according to his requirements/service provider.

According to [110], a distributed system is a system whose components are located on different networked computers, which communicate and coordinate their actions by passing messages to one another. Distributed computing is used in many areas, for example, management of banking transactions, air traffic control, supercomputing, distributed file storage system. One of the most important use cases of distributed computing architectures is the execution of complex computing operations, which cannot be achieved using only one computer, or which takes a longer time when executed on only one computer. This is why it is important to execute this work on a parallel architecture sustained by a significant number of computing resources.

In the general case, a distributed computing architecture is composed of mainly two actors, a service provider and service consumers (users). Each of them looks to benefit as much as possible from the computing infrastructure. To avoid a conflict of interest between these actors, negotiation must be done before starting work, both providers and users must sign a contract before engaging. The contract is called Service Level Agreement (SLA). During the SLA negotiation, a trade-off between both providers and users must be ensured. Then, many parameters can be discussed. Usually, these parameters are called Quality of Service (QoS) parameters.

The facilities proposed through the services hosted in these kinds of infrastructures are more and more various and thereby increasing the number of users. The complexity of such an infrastructure raises many issues regarding the functioning of the services, especially to ensure reliable and efficient access to users. Indeed, computing resource providers must keep their infrastructure running smoothly and propose a high level of QoS in order to fit users' expectations. These considerations force the computing service providers to analyze the characteristics of their systems in order to manage and improve them to find the best trade-off between their own interests and the level of quality of service proposed to the users.

In the distributed computing area, two main notions need to be known before talking about scheduling and resource management solutions. The first one is Resources, and the second one is Work. Usually, these two elements are collaborative, which means that each one of them must know the characteristics of the other. More in detail, a resource must know its characteristics to be able to accept or refuse a work or a part work, while a task must now its requirements to be able to select the set of candidate resources that can be used for future allocation for their execution. Usually, the informations can be managed through a resource management solution.

From a resource point of view, the term Resource is used to refer to any virtual or physical equipment that can be shared, and used by a resource management solution.

Resources can be a virtual or physical machine equipped with an operating system, a virtual network equipment, or a shared memory.

The term work refers to a program composed of a set of tasks, each of which represents a program unit that cannot be subdivided into subprograms. In a given work, the necessary time to finish the last task is called *makespan*. In order to ensure the execution of a work in a distributed environment, a set of resources must be allocated, then, each task of the work must be assigned to one of these allocated resources. The latter are often heterogeneous, which means that the execution time of a task changes from a resource to another. Subsequently, the *makespan* changes according to the user mapping. Therefore and in order to ensure a better mapping of the set of tasks to the set of allocated resources, a power scheduler must be used.

Before finding the mapping of tasks, a scheduler needs to be supported by a set of informations about the resources that compose the architecture, and by a set of informations about the tasks to be executed. After finishing the mapping, the scheduler sends the message. It then needs to get help to execute and measure QoS parameters. One of the most important research issues raised is how to smartly manage those resources.

From a provider's point of view, resource management is a critical aspect as it can have a huge impact on the functioning of its whole infrastructure. In most cases, a computing service can be seen as a set of tasks that have to be scheduled over a set of available resources in the computing infrastructure. As each resource is a whole or a part of a computing service, any modification of the task allocation has to be done without impacting the users' experience while trying to optimize the global behavior of the services. Also, several parameters (objectives), likes response time, resource usage, cost, power consumption, load balancing, etc., need to be taken into consideration when designing and evaluating a task scheduling solution.

Scheduling has a specific role, which consists in mapping a set of tasks to a set of resources. Meanwhile, resource management consists in doing more actions, like resource discovery, request collector, resource allocation, and resource monitoring. Resource management can be changed according to the used design. As an input of the scheduler, many values can be needed before running a scheduler process, for example, the estimated execution time, the estimated cost, the estimated consumed energy. The output of the schedule consists in affecting of the set of tasks to the set of resources. Also, a set of characteristics that concern the work need to be defined, for example, new tasks can be introduced during the execution processes, some tasks have higher priorities than others,.. etc. With regard to the scheduler also, it is

necessary to define some characteristics, for example, can we change the result during the execution of the resulting mapping ...etc.

Research Problems

Today's distributed computing infrastructures, including Grid, Cloud, Fog and other distributed system paradigms, are characterized by great difficulty in managing their resources due to a few characteristics, such as the high level of resources' heterogeneity, instability the environment, where resources can join or leave the infrastructure at any time and tasks can arrive sporadically on any site and at any time, and the momentum of change of these infrastructures in terms of the appearance of new architectures or changes in existing architectures. These factors have given rise to other problems. In our thesis, we are interested in a sub-part of these problems that we will present in the remainder of this subsection.

The problem of task scheduling in distributed computing environments remains one of the most important challenges. In general, scheduling problems are classified as Non-deterministic Polynomial-time Hardness (NP-Hard) [37], which means that there is no known general solution for which the optimal value of the objective(s) to be optimized can be obtained in a time that is polynomial with the size of the problem. Consequently, a significant number of approaches have been proposed in the literature, which aims at finding a possible mapping while getting the optimal value of the objective(s) to be optimized within a reasonable time. Fortunately, the new distributed architectures, characterized by their powers and their varieties, always give the possibility to think about proposing new algorithms. Task scheduling in distributed computing systems is our first problem treated in our thesis work.

The second issue addressed is closely related to the first one presented above. It is motivated by the fact that there are a large number of approaches of task scheduling already proposed in the literature. These approaches, although they often produce interesting results, are almost always carried out only by simulation and are therefore, unfortunately, never deployed and tested in real conditions. This is mainly due to the following reasons:

- The approach was not necessarily designed to be deployed in real conditions,
- There is a huge time cost to spend to make it run smoothly on Cloud or Fog platform,
- Even if some approaches can be run on real distributed systems, the portability and reproducibility of the experimentation are another hard point to face with.

To the best of our knowledge, now has addressed simultaneously all the points mentioned above, which motivated us to work on those issues and to propose a solution that can contribute to resolve the second point. Indeed, our smart solution tries to propose a clean and reproducible scheduling solution regardless the distributed system involved.

Objectives

In order to contribute to the resolution of the problems mentioned in Section , we have divided our objectives into two parts, the first one is related to the task scheduling problem, while the second one is related to the resource management issue.

The first objective of our work consists in contributing in the field of task scheduling in distributed computing systems, by proposing solutions that can minimize the total response time as much as possible. The proposed solutions must take into account the scalability factor (extensibility) as well as the great heterogeneity of today's distributed systems. Also, the time of finding the mapping must be considered, and reduced as much as possible.

Our second objective is related to the first one, it consists in proposing a solution for resource management that can exploit our proposed algorithms, or any other task scheduling algorithm. This solution must offer the possibility to make an intelligent management of the resources of such a distributed computing system. This solution must also make it possible to do a "From theoretical to real deployment", i.e. an automatic switch from the evaluation phase to the deployment phase. Therefore, the tool should be easily adaptable to different kinds of distributed environments, take into account the great heterogeneity of resources, and support the scalability of the use distributed system.

In more detail, our solution should offer to the research and development communities a reliable solution, which can help them understand the behavior of the used architecture, and which can also evaluate such scheduling algorithms in theoretical values first, predict the behavior of the platform before deployment, and then deploy the best found solution in a real environment in a third step. While studying the behavior of these algorithms during a real execution, our goal is not only to study the total response time but also other parameters of QoS that we considered important, such as cost, energy consumption, and resource usage.

Contributions

With regards to scheduling, we have proposed two algorithms that consider minimizing the total response time. The first is a simple heuristic called MMin, and the second is an evolutionary approach called Inter-Resources Collaboration Heuristic (InterRC).

The *InterRC* heuristic belongs to the family of evolutionary approaches which are considered as an important way to solve the scheduling problem. Usually, in an evolutionary approach, the value of the objective(s) to be optimized is improved over time, by starting from a solution called *initial solution* which is then improved through iterations until reaching one or some conditions called *end conditions*. In this way, *InterRC* tries to find the best match between a set of tasks and a set of heterogeneous resources in a distributed environment, with the goal of minimizing the total response time. In the *InterRC* approach, we use a new concept called inter collaboration. As the name implies, *InterRC* is trying to evolve from one solution to a better solution. To do this, each time we switch from one solution to another, the resources try to exchange tasks with each other and test whether this exchange improves the total response time. In the positive case, this exchange is carried out and in the opposite case, the exchange does not take place. *InterRC* uses certain operators and stopping conditions that will be defined in detail in this manuscript.

For *MMin*, since simple heuristics are important in terms of fast execution time, we propose in the second contribution a simple heuristic called *MMin*, The latter inherits the advantages of *Max-Min* and *Min-Min*. To avoid the disadvantages of *Max-Min* and *Min-Min*, the proposed solution is executed in n iterations, where n represents the total number of tasks to be scheduled, then, in each iteration, *MMin* calls one of these two solutions depending on the overall state of the system.

The third contribution consists in proposing a solution that allows to manage the resources of the distributed computing infrastructure, and in order to meet our second objective by doing a "From theoretical to real deployment", we proposed a framework called Modular scheduling Framework for Heterogeneous System (MFHS). The latter is composed of several modules, each of which has its own role: Resource Discovery, Request Collection, Scheduling, Resource Allocation, and Behavior Study. These modules are easily adaptable in any heterogeneous distributed environment. The validation of our framework has been done in three different environments which are: a Personal Computer (PC) (using MFHS as a simulator), an OpenStack-based Cloud located at the Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS)

of the Centre National de la Recherche Scientifique (CNRS) (LAAS-CNRS, Toulouse, France), and an Emulab-based test platform located at the Centre de Recherche sur l'Information Scientifique et Technique (CERIST, Algérie). The aim of our experiments is to show the proper functioning of MFHS on the one hand, and to show the efficiency of our *MMin* algorithm compared to two other algorithms on the other hand, in terms of total response time. In addition, MFHS allows measuring the cost, the consumed energy, as well as the load balancing between resources.

MFHS is composed of several modules, each with its own role: *Resource discovery*, *Request Collector*, *Scheduling*, *Resource Allocation* and *Behavior Study*. The extensive experimentations exposed in Chapter V, were carried on in order to validate *MFHS* using our proposed *MMin*, *Max-Min*, and *Min-Min* algorithms. The validation is done by both using theoretical analysis and real deployment on a research Cloud platform.

The third contribution can be, briefly, expressed as follows:

- Definition of a Cloud Computing model dedicated to the off-line scheduling of independent, non-preemptive and fixed priority tasks,
- Proposition of a framework that can be used in both virtual and real distributed computing platforms, which integrates a set of modules, and is easily adaptable in any heterogeneous distributed environment,
- Highlighting of the reliability of the proposed approach using three different environments and expose wide experimentation using the experimental Cloud based on *OpenStack* which compares the efficiency of the proposed *MMin* against two others algorithms in terms of QoS metrics.
- Validation of *MFHS* as a simulator, while using a theoretical data.
- Basic evaluation of *MFHS* using *Emulab* testbed,
- Exposition of wide experimentations using *MFHS* in a real Cloud environment based on *OpenStack* which compare the efficiency of a heuristic and algorithms through the computation of a set of metrics and parameters.

Thesis Organization

The remainder of the thesis is organized as follows:

In Chapter I, we discuss the background that is related to the context of our work. Firstly, we present the most widely considered characteristics of schedulers and tasks. Then, we talk about a set of distributed architectures, especially recent ones. Finally,

we present different QoS parameters. This chapter allows understanding the ones that follow it.

In Chapter II, we introduce the related work. Two sub-sections are presented in this Chapter. The first one consists in describing the task scheduling algorithms existing in the literature, that we have classified into two categories, which are non-evolutionary, and evolutionary approaches. Then, in the second sub-section, we present a set of evaluation tools, allowing to evaluate such an algorithm. More in detail, this second part presents the simulators of distributed environments in a first time, and the frameworks oriented resource management in a second time.

In Chapter III, which is mainly derived from our publication [62], we present in detail our framework *MFHS*. In this chapter, the architectural model that we have adopted by our framework is presented. Then, the different modules, as well as the different formulas used to compute various QoS parameters are discussed.

In Chapter IV, we describe in detail our two proposed algorithms, which are mainly inspired by our two publications [61], and [62]. The first algorithm is a deterministic heuristic called *MMin* which is presented in our publication [62]. *MMin* combines the two well-known heuristics *Max-Min* and *Min-Min*. The second algorithm is another evolutionary approach that we presented in our paper [62]. Remember that the objective of our two proposals is to optimize (minimize) the total response time.

In Chapter V, we present in detail the evaluation of our proposals, a presentation of InterRC using a simulator is described in the first part, then, various experiments that we have carried out using *MMin* and MFHS are discussed in the second part. The latter is quite large compared to the first one, because it brings together two objectives, on the one hand the well-functioning of our MFHS framework, and on the other hand the efficiency of our *MMin* algorithm in terms of total response time. The experiments done in the second part were conducted on three different environments, the first one is a simple PC where MFHS was used as a simulator, the second one is an Emulab environment located at CERIST and the third one is a Cloud environment located at LAAS. Four QoS parameters were considered in each experiment, namely, the total response time, cost, energy consumed, and resource usage.

In Chapter 5.5, which is the last one in our thesis, we start with a discussion of the different results that we have obtained during our various experimentations, then we finished with a conclusion and with a presentation of several perspectives that can follow our work.

List of Publications

As results, two papers are published. The first one exposes our evolutionary solution of task scheduling, which is published in the "Mendel Soft Computing journal" after its presentation in the 25th International Conference on Soft Computing. Then, the seconde one is published in the journal "Software: Practice and Experience", it exposes our framework, and our heuristic MMin.

Our two publications are:

1. Abdelhamid Khiat and Abdelkamel Tari. Interrc: An inter-resources collaboration heuristic for scheduling independent tasks on heterogeneous distributed environments. In MENDEL, volume 25, pages 179–188, 2019.

2. Abdelhamid Khiat, Abdelkamel Tari, and Tom Guérout. MFHS: A modular scheduling framework for heterogeneous system. Software: Practice and Experience, 50(8):1463–1497, 2020. doi: 10.1002/spe.2827.

CHAPTER I

Background

1.1 Introduction

Before giving details about the main subjects of this thesis, which are the resource management and scheduling in large scale distributed systems, we will start by explaining a set of key-concepts which represent the background needed to understand our contributions.

In this chapter, we will present the background. We start by giving the most important properties related to task scheduling, then we discuss a set of distributed environments which can be used to achieve a parallel computing, and finally, we present different QoS parameters, where one or more of them can be considered when proposing such a solution for task scheduling or resource management oriented to distributed environment.

1.2 Scheduler Characteristics

Scheduler invocation There are mainly two ways to invoke a scheduling process: event-guided invocation [18] and time-guided invocation [69]. In the first case, the invocation of the scheduler is decided following an event such as task arrival, or change of a resource's state. Whereas in the second case the invocation time of the scheduler is defined beforehand and does not depend on any event. It is common to use the term *on-line* scheduling to mean event-invoked scheduler, and the term *off-line* scheduling is used for a time invoked scheduler.

Scheduler nature Two types of scheduling can be distinguished. Dynamic scheduling and Static scheduling. According to [107], in a static scheduling case, the scheduler has complete view of the tasks set and its constraints. such as, precedences

constraints, execution time, preemption, priority, etc. In contrast, a dynamic scheduler has complete knowledge of the currently active set of tasks, but new arrivals may occur in the future, not known to the algorithm at the time it is scheduling the current set. The schedule therefore change over time.

Determinisity Algorithms can be classified as either deterministic or non-deterministic. In a deterministic algorithm, the results are identical even if the execution of this algorithm is repeated many times. Conversely, in a non-deterministic algorithm, the results vary at each execution of the algorithm.

1.3 Task Caracteristics

Priority It defines whether some tasks of the whole set taken into account have to be scheduled before the other or allowed to be delayed. When all tasks to be scheduled have the same priority value, the scheduler is said without priority and all the tasks have the same probability of being started first. Contrary, for task scheduling with priority, the value of this latter can vary between an interval which defined the tasks that have a higher priority than the other. In general, a higher priority leads to have a higher probability of being schedule firstly.

In priority scheduling, two cases are possible: dynamic or static priority. In the case of dynamic priority scheduling, the priorities are computed during the execution of the tasks, whereas, in the static priority scheduling, the values of the priorities are fixed and can not be changed during the tasks execution.

Deadline A task can have a constraint on the execution end date, meaning that the task completion time must be lower than a given time. In this case, the scheduling approach is said to be time-constrained (with deadline). Contrarily, when the task completion time is not subjects to a deadline, the scheduling approach is not time-constrained.

Preemption We say that a task is preemptible when it is allowed to suspend its execution. The suspension is generally done to execute another task with higher priority than the suspended one. Meanwhile, a non-preemptible task is a task whose suspension is not allowed in any case which means since its execution has been started the task has to be completed.

Dependence This characteristic allows to define the relationship between tasks. Two cases are possible: dependent tasks or independent tasks. In the case of independent tasks, there is no precedence constraint between tasks, i.e. each task can be started without waiting for another. Whereas in the case of dependent tasks, there are precedence constraints between the tasks, where, each task must respect the dependency constraints and wait for the completion of all preceding tasks before it is started. In this second case, the term workflow is widely used and a Directed Acyclic Graph (DAG) is also used to model a tasks scheduling problem using dependencies constraints.

Tardiness In a time-constrained scheduling approach, the tardiness of a given task represents the delay penalty of this task. It is defined as the time between the due-time (deadline) and the real completion time of the corresponding tasks. It is significant only if a task is completed after its due-time [94].

Periodicty It is possible to classify the tasks in terms of their appearance into three main categories: periodic, sporadic, and aperiodic tasks. A periodic task is a task that repeats after a certain fixed time interval (clock interrupts). A sporadic task is a task that recurs at random instants. An aperiodic task is in many ways similar to a sporadic task, except that, in case of an aperiodic task, the minimum separation time between two consecutive instances can be 0. That is, two or more instances of an aperiodic task might occur at the same instant.

1.4 Distributed Architectures

Since their, distributed systems never stopped evolving, especially with the advent of new types of resources that can be solicited to participate to a distributed computing, like smartphones, tablets, or any other network equipments that possess a computing equipment (processor). Innovation in this kind of computing architectures leads to the apparition of new ones, and also increases the importance of these paradigms of computing. In this chapter, we aim to review the most important distributed computing paradigms and the main similarities and differences between them, in particular, the ones that can be supported by our proposed solutions.

Cluster the concept of Cluster has emerged in the early 1970s. A cluster was organized in the following way: a front-end machine that allows the user to submit

tasks, and a set of storage and computing nodes that allow the user to execute the tasks. The nodes are linked together by a local area network. A backup machine allows to store the current job while the cluster is processing another user's jobs. This type of clusters has several drawbacks, in particular the need to expand their capacities without turning them off. Beyond a certain size, they will pose a problem of space in buildings, electricity supply, or air conditioning. As another drawback, there is the fact the users will have to share the same environment. In other words, there cannot be, for example, users who use their own operating system: users must work with the programs and libraries installed on the cluster's machines.

Today, clusters are still used for High-Performance Computing (HPC). The Top500[1] is a ranking listing the 500 most powerful clusters in the world. For example, in November 2019, the top-ranked cluster in the ranking is the Summit[2] supercomputer developed by IBM, which provides a Theoretical Peak (Rpeak) of 200,795 $TFlop/s$.

At the same time, the organization of the clusters has changed: clusters have been made usable as a single computer with "single-image systems" [82]. A low-level overlay allows the merging of all nodes. For example, when the user invokes the *ps* command, system calls are modified to return a list of all processes running on all nodes of the cluster. Such implementations have been proposed by OpenSSI [3] or open [4]. The disadvantage of this kind of systems is the cumbersome maintenance. Each addition to the operating system kernel leads to a change in the software.

Nowadays, the amount of data to be processed increases significantly and in a BigData context, transferring the data to the computing node is no longer possible. It was therefore considered to move the processing to the servers storing the data rather than the other way around. In this way, the computing and storage nodes were merged. In this area, implementations such as Hadoop [118] or over-layers such as Spark [129] or Flint [100] are used.

Grid computing The concept of Grid emerged in the 1990s. A grid can be seen as a set of clusters geographically distributed over a territory and interconnected (either by a dedicated network or via the Internet). Usually, this interconnection network has a much higher latency than the latency of the internal network of each cluster. Those high levels of latency bring many difficulties, for example, for the sharing of

[1]https://www.top500.org/lists/2019/11/
[2]https://www.top500.org/system/179397
[3]http://openssi.org/
[4]Mosixhttp://openmosix.sourceforge.net/

data between different sites. Accessing data located in another cluster of the grid becomes an expensive operation. File systems such as Xtreemfs have been invented for this particular use case [51].

A major concept that makes the grid different from other distributed architectures is the use of Virtual Organization (VO). The latter allows to organize the whole set of grid users into virtual organizations that work on the same resources; which are not, generally, accessible by the other VOs. The shared resources can be data, material devices, processing software, etc.

Cloud Computing Cloud computing is composed of a very large number of servers distributed in a limited number of data centers. Virtualization technologies is the main used technology, which allows these nearly infinite resources to be allocated on demand by users [130].

Cloud computing providers can provide services at different levels of abstraction such as: Software as a Service (SaaS) which consists in hosting and maintaining an entire application for a user, Infrastructure as a Service (IaaS) which consists in providing only virtual machines that the user can configure himself, or Platform as a Service (PaaS) which provides both the hardware and operating systems ...etc.

SaaS, IaaS, and PaaS are the most known abstraction levels, but there exist other levels, like Data as a Service (DaaS) that is considered as a subset of SaaS. Function as a Service (FaaS) is even simpler than PaaS. As suggested by its name, it is based on the functions which can be triggered by a given event. Backend As A Service (BaaS) differs from PaaS. It provides the tools and services necessary for the development and operation of the applications, allowing IT departments to avoid investing in the development and management of their back-end infrastructures.

IoT Due to their centralized architecture, Cloud Computing infrastructures are not adapted to the needs of the Internet of Things (IoT). Firstly, the long-distance between connected objects and data centers delays responses to users, because of the low latency. This makes its use incompatible with a large number of use cases in which the response time is critical [10]. Some authors go so far as to say that the latency to reach a server located on a Cloud Computing site is high and unpredictable [130]. The second reason is that the number of existing data centers can not keep up with an ever-increasing number of connected objects, considering the high number of requests and network load [130]. To give an order of magnitude, Shi et al [102] indicated that in 2019, the connected objects will produce 500 zetta-bytes of data, whereas it is

estimated that the amount of data produced by Cloud infrastructures can currently only carry 10.4 Zo (we are not able to check this value under the day of writing this thesis).

IoT has been proposed in order to avoid the Cloud weaknesses. According to ITU[5], IoT is defined as "global infrastructure for the information society, which provides advanced services by interconnecting objects (physical or virtual) using existing or evolving inter-operable information and communication technologies".

Edge The general idea of the Edge solution consists in carrying out the computation, no longer on the Cloud Computing infrastructure but on servers located at the edge of the network, as close as possible to the users and connected objects. To do this, servers are deployed close to the users, at the edge of the network [57] [35]; [102]. In the case of a cellular network, servers are deployed near each "base station". Users then use the resources provided by the servers located in the cell to which they are connected. These servers perform calculations that require low response times and pass on those that require higher resources to the Cloud. Such an architecture is also useful when storing data: the user queries a server located at the edge of the network. If the latter does not have sufficient resources, the request is transmitted to the Cloud infrastructure. When the data is returned by the Cloud, it is cached so that future requests can be answered directly. Servers located at the edge of the network have fewer resources than those in the Cloud, but sufficient resources to perform some operations that require low latency.

Extreme Edge Computing is a specific case of Edge computing. The "Extreme Edge Computing" model has the same objective as Edge Computing: to process requests as close as possible to the users in order to limit the demands on the Cloud infrastructure. However, the implementation is different: instead of placing new servers close to users, users pool and share resources using peer-to-peer (P2P) or ad-hoc protocols [64]. Calculations and storage are therefore performed on the devices (connected objects) themselves. The differences between these different infrastructure models are summarized in details by Yi et al [127].

Fog Computing For several authors, "Mobile Cloud Computing" type infrastructures cannot cope with the mobility constraints of connected objects [126] [79]. Thus on small cells or when the connected object has a very high mobility, the server on which the calculations are deported will change permanently. This change will be

[5]https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=11559&lang=en

the cause of high latencies when submitting a computation. "Low Cloud", "Fog" or "Fog Computing infrastructures" were proposed by Cisco[6] in 2012 [10] and are today supported by many manufacturers. The proposed architecture consists in deploying, not isolated servers close to the users but, small data centers distributed in different sites located at the periphery of the network. These data centers typically include about ten servers and provide computing and storage resources to clients located at the edge of the network. Because of their larger size compared to Edge Computing, Fog's sites support greater user mobility while improving response times compared to a cloud infrastructure. In order to simplify deployment, it has been proposed to place the servers in points of presence on the Internet [71]. Each device accesses a server located in its immediate environment. This server performs some computing locally and delegates those that are too costly to another server located a little further away from the end user. In this way, the computations are distributed throughout the infrastructure. The servers close to the users take care of the inexpensive but low response time processing, up to the Cloud Computing which performs the very resource-intensive calculations but for which the response times are less important.

Fog infrastructures can also be hierarchical, with a Cloud computing architecture at the top of the hierarchy [11] [33] [14]. The general idea is that, the more a site is in the top of Fog hierarchy, the more the provided resources are important. On the other hand, the latency to reach this site is high. The Cloud provides almost unlimited storage and computing capacity at high latency, while the Fog site closest to the user provides very low latency, low computing and storage capacity. The advantage of such an architecture is that when a Fog site needs to access data or perform computations in the cloud, the cloud infrastructure is not directly affected. Many sites are traversed, allowing data to be cached, aggregated, and even some portions of the computations to be performed [1]. As a result, the Cloud receives much less data than when it is used directly, allowing a better scalability. In some cases, all the computing is performed before the Cloud infrastructure is reached, reducing calculation time. The general idea of the Fog is to be able to perform Big Data calculations with latency constraints [9]. Such an architecture has other advantages, especially from a privacy and security point of view [108] [128]. Having the user's data on a server located close to the user limits the number of potential attackers that may be encountered the way. Similarly, each server only manages the data of a small number of users, which limits the impact in case one of them is compromised. Finally, other authors evaluate the potential energy gain by redirecting queries to the Fog site or data center with the

[6]https://www.cisco.com/

lowest carbon footprint at the time [55] [34].

1.5 QoS parameters

QoS (Quality of Service) is an important criterion that must be considered when designing solutions for task scheduling or resource management, which aims to ensure a good execution of users's job in order to satisfy the architecture users. For this, several metrics can be considered. In this section, we will present a set of such metrics and give a brief description of each one of them. These criteria help also to evaluate the quality of a scheduling solution, although it should be noted that this quality depends not only on the scheduler policy, but also, on the state of the used computing platform. In chapter III, we will take some of the parameters presented here, especially the ones that we have used in our solutions, then we give more details about the formulas used to calculate them. QoS parameters can vary from one field to another. In the following, we will focus on the QoS parameters that are the most relevant in the field of distributed computing, especially those that have a direct influence on the system behavior

Makespan One of the most commonly used measures in the evaluation of scheduling algorithms is the Total Completion Time or Makespan. The makespan is the difference between the time of job submission and the time at which result is obtained. In other words, it is the completion date of the last task. In general, the makespan depends of the task execution time and the time taken by the transfer of the data over the network. The execution time depends on both workload and machine performance. The network transmission time depends both on the latency of the network and the size of transmitted data.

Cost As a result of the market-oriented feature of today's services, most providers of distributed computing platforms have set prices for the use of their services. The cost is generally set in relation to the transfer of a unit of data (e.g. per MB) between the resources used and the price for processing per unit of time (e.g. per hour). Other parameters are also considered such that the quality of the resource used, i.e. the unit cost of using a resource is higher when it has a low probability of having a failure than when it has high probability.

Reliability Reliability represents the probability that the job will be fully executed successfully, without any resource fault. Several models are available, such as the one described in [114], which is an intrinsic property of the resource.

Consumed Energy This parameter becomes more and more important, which imposes its consideration when proposing a scheduling or resource management solution for such a distributed architecture, independently of its nature. According to planetoscope [7], world energy consumption represents 428 tonnes of oil equivalent every second (meter) or 13,511.2 Million tonnes of oil equivalent (Mtoe) per year. Fossil fuels still account for 81% of world consumption.

Load balancing This parameter is considered as one of the key issues: it is necessary to distribute the workload between several nodes to ensure that no node is overloaded compared with others. Load balancing must be ensured while optimizing other parameters such as makespan and resource utilization.

Resource utilization Is the efficient and effective use of the set of available resources. In this way, all resources allocated for such a job need to be used as much as possible to avoid an additional consumed energy and cost .... This parameter becomes more important, when the computing providers compute the cost per time and not per Central Processing Unit (CPU) time utilization.

1.6 Conclusion

In this chapter, we have described different notions, which represent the background necessary for the user to introduce our own work. First, we have presented many notions concerning the task scheduling part. We also presented different properties related to the tasks, and different parameters concerning the scheduler. Secondly, we have outlined several parameters of QoS that allow us to analyze the behavior of task scheduling approaches in a distributed computing architecture. Finally, we have discussed different kinds of distributed computing architectures, which are required to understand the philosophy of these architectures.

All presented material is used in the remainder of this thesis. In the following chapter, we analyze a set of scheduling heuristics. Meanwhile, the presented heuris-

[7] https://www.planetoscope.com/Source-d-energie/229-consommation-mondiale-d-energie-en-tep-.html

tics are classified into two categories, evolutionary heuristics and non-evolutionary heuristics.

CHAPTER II

Related Work

2.1 Introduction

In this chapter, we present our related work. For a more clear presentation, we have divided this chapter into two parts. In the first one, we discuss task scheduling whereas in the second, we cover resource management. In the first subsection, we focus on different task scheduling heuristics, especially, that are tightly related to our proposed scheduling solutions. Then, the second subsection which covers resource management is further presented as two subsections: simulation tools that allows to simulate a distributed computing environment, and frameworks that can be deployed in real world distributed architectures.

2.2 Scheduling Algorithms

The complexity of the task scheduling problem in distributed environments when it comes to find the optimal *makespan* is NP-Hard in general [37]. Consequently, there is a lack of solutions that can find the optimal *makespan* in a reasonable time, especially when the problem size increases. An important number of approximate approaches that address this problem have been proposed in the literature. These approaches aim to find a solution as near as possible to the optimal solution in a reasonable time.

A set of non exhaustive works will be presented in the remainder of this section. The presentation is divided into two subsections. In the first one, a set of non-evolutionary heuristics, characterized by their low execution time complexity, will be presented, including a set of heuristics based on both Max-Min and Min-Min. Then, in the second one, some evolutionary approaches will be introduced, which are usually characterized by a good result in terms of objectives to be optimized.

2.2.1 Non-evolutionary Approaches

2.2.1.1 Conventional Heuristics

Max-Min [30] algorithm consists of executing a set of iterations. The iteration process consists of selecting the task that has the biggest completion time, and then affects it to the resource that gives the minimum execution time. It subsequently repeats this process until the end of scheduling all tasks. After each iteration, the completion time is updated for each task that is not yet executed.

In the *Min-Min* [30] scheduling process, an iteration consists of selecting the resource that has the minimum value of completion time, then, the task that has the minimum execution time on this resource is selected. Similarly for *Max-Min*, after each iteration the completion time is updated for all tasks not yet mapped.

Min-Max [53] heuristic is similar to *Max-Min* and *Min-Min* approaches, it schedules one task in each iteration, until the scheduling of all tasks. At each iteration, the minimum completion times of all unassigned tasks over all available resources are computed. Then, for each unassigned task, the ratio of its minimum execution time on all resources to the execution time on the processor that resulted to the minimum completion time is computed. The task that has the highest value of this ratio is removed from the list of unassigned tasks and scheduled to the resource that gives the minimum completion time.

Sufferage algorithm [77] executed in iterations where each iteration is composed of two processes. The first one consists of computing for each task a value called $sufer$, which represents the difference between the first and the second minimum execution time of the concerned task. While the second one allows affecting the task with maximum $suffer$ to the resource that gives the minimum completion time. These two processes are repeated until the end of the assignment of all tasks.

Round Robin (RR) algorithm is a simple heuristic with low complexity, which remains largely used in a significant number of algorithms, particularly, in the real deployed algorithms. the RR algorithm affects the first task found in the set of not affected tasks to the first available found resource, until there are no remaining tasks.

An algorithm called Relative Cost algorithm (RC) was proposed in [122]. Relative Cost (RC) utilizes an indicator called Relative Cost (RC). According to authors, RC retains the advantages of the *Min-Min* algorithm regarding *makespan*, and efficiently balances the load. The task and resource that are selected in each iteration is based on two quantities: the static relative cost and the dynamic relative cost. The static relative cost is computed once at the start of the algorithm as rate between the

execution time of this task on this resource to the average of its execution time on all available resources. The dynamic relative cost is computed before each task is scheduled as the rate of the completion time of the task on the resource to the average of its completion time on all available resources.

2.2.1.2 Heuristics based on both Max-Min and Min-Min

The heuristics that are presented in section 2.2.1.1 are considered as conventional heuristics. Therefore, an important number of heuristics is based on the presented heuristics. In the following, we expose the most notable ones. Looking to the huge number of this kind of heuristics that exist in the literature, we will focus only on the ones based on Min-Min and Max-Min. This choice is made because one of our contributions combines the two algorithms.

In the litterature [75] [27], the complexity of *Min-Min* and *Max-Min* is known to be $\mathcal{O}(mn^2)$, where n represents the number of tasks and m the number of resources. Nevertheless, an implementation of [109] that switches from the original task-oriented view to a processor, reduces the complexity of *Min-Min* to $\mathcal{O}(mn \log n)$.

In [88], the authors have proposed a hybridization of *Min-Min* and *Max-Min*. The proposed algorithm, called RASA, starts by counting the number of available resources. If this number is even, the algorithm calls the *Min-Min* algorithm, if not, it calls *Max-Min* algorithm. Then, the execution will be alternatively between these two algorithms. In [81], the authors have proposed to combine RASA with *Min-Min* for scheduling tasks in a Cloud environment by dynamically choosing in each iteration the adequate solution to be used (i.e RASA, Min-Min) to be used. In both proposals, the authors looks to optimize the total makespan.

Another hybridization of *Min-Min* and *Max-Min* is proposed in [31]. The proposed algorithm selects in each iteration an algorithm (*Min-Min* or *Max-Min*) to be executed. The selection is based on the computation of the position of the first task in the meta-task in which the distance between its completion time and its successor's completion time is greater than a value called *sd* (standard deviation). If this position is less than $n/2$ (task located in the first half of the meta-task) or if *sd* is less than a predefined threshold, then the *Min-Min* algorithm is selected. Otherwise, the *Max-Min* algorithm is selected.

Another approach based on the statistical concept called *asymmetry coefficient* is proposed in [87]. According to the author, *Min-Min* gives a good total response time and the *Max-Min* gives a better machine utilization rate in terms of load balancing

between resources. Besides, *Min-Min* gives poor results for a positive skewness coefficient because the majority of tasks are of small size. The skewness *asymmetry coefficient* is computed at each iteration. If the skewness is less than zero, then the *Max-Min* algorithm is used, otherwise the *Min-Min* algorithm is executed. This new algorithm shows a better load balancing and total response time compared with two classical algorithms.

In [73], the authors have proposed a heuristic that looks to minimize the makespan of a job which is executed on geo-distributed data-centers. The proposed heuristic called *Min-Max-Min*, extends *Max-Min* algorithm. In the proposed heuristic the task which has the shortest expected completion time has the priority to be selected first, while the details of the proposed heuristic were not given in the article.

Usually, the algorithms based on *Max-Min* and *Min-Min* consider only one parameter related to the execution time. Whereas an improved *Max-Min* algorithm was proposed in [58] which considers additional job's characteristics like size, outline, payload ratio, and available storage.

Other propositions look to improve one of the two algorithms, as for example in [28]. The authors made only one change to the original *Max-Min* algorithm. The latter begins with the assignment of the task that has a maximum completion time to the resource that offers a minimum execution time, whereas, in the proposed version the scheduler starts by the assignment of the task that has a maximum completion time to the resource that offers a minimum completion time (always by a recursive execution of the algorithm). Thereafter, Bhoi et al. proposes a single change to the previous algorithm([28]). The basic idea of this new algorithm is to assign the tasks that their completion time's value is the nearest to the average of the completion times of the other tasks (not the greatest).

An algorithm based on *Min-Min* was proposed in [4] where the processing and the data transfer speed are considered. The principle of the proposed algorithm consists to divide the set of resources into four categories: low/high data transfer rate and high/low processing speed. Afterward, for each task, the algorithm computes the value of CCR (communication to ration computation) to determine the nature of the job. The latter can be in high/low QoS in terms of CN (Communication Link) or high/low QoS in terms of CPU (CPU speed). According to the job nature and to the resource category, the algorithm determines the set of candidate resources for the job's execution. Then, *Min-Min* algorithm is executed for each job on the set of candidate resources to select the appropriate resource to execute the job.

In [60], a two phases scheduling algorithm is proposed. The first phase consists

to call the standard *Min-Min*. In a second phase, the rescheduling is executed by ordering the tasks scheduled on the resource giving the makespan by decreasing order. It then finds another resource that can execute one of those tasks and reproduce the expected makespan.

Max-Min and *Min-Min* are also used to optimize other parameters like load balancing. In [40], the authors propose a load balancing solution in addition to the total time response optimization. The idea consists of executing the algorithm *Max-Min* in two phases: the first one consists of executing the classical *Max-Min* and the task is rescheduled in the second phase to achieve the maximum load balancing. To do this, the algorithm compares the MCT (Minimum Completion Time) of each task Ti with the response time produced by the standard *Max-Min* algorithm (after sorting tasks in ascending order).

In [68], expanded *Max-Min* (*Expa-Max-Min*) algorithm is proposed. Based on *Max-Min*, it aims to reduce the cost and balance load by effectively give equal opportunity to both cloudlets with maximum and minimum execution time to be scheduled first.

In [116], the authors have proposed an application of the *Max–Min* Ant System in mobile Cloud computing. The authors present a local mobile Cloud model with a detailed application scheduling structure for the first time. Then, a scheduling algorithm for the presented model is presented. The presented algorithm is based on *Max–Min* ant system called *MMAS*.

In [50], based on the original *Max-Min*, the authors introduce a machine learning approach to improve the completion time in a Cloud environment. The proposed algorithm is called *MMSIA*. After clustering the size of requests and clustering utilization percent of the set of Virtual Machine (VM), *MMSIA* uses the "learned learning" machine learning approach. Then it assigns the largest cluster requests to the VM with the least utilization percent.

2.2.1.3 Other Non-evolutionary Approaches

In this section, we present some heuristics that cannot be considered as conventional.

LSufferage proposed in [41] is inspired by *Sufferage*. In the *LSufferage* algorithm, a static descending ordered list is generated for each possessor p. Each element of the generated lists contains the task identifier, and an associated value. The latter is obtained by computing the ratio between the maximum execution time of the

concerned tasks T and its execution time on p if the execution time of T on p is not the maximum execution time, otherwise, the value is calculated by the division of the execution time on the second fastest possessor on the maximum execution time (p). Finally, the scheduler uses these values to schedule each task to the processor according to their priority (computed ratios).

In [59], a two-phases scheduling algorithm is proposed. In the first phase, the traditional Min-Min algorithm is executed. In the second phase, the tasks are rescheduled to improve the use of the improve the resources utilization.

In [54], a two level load balancing approach is proposed, which aims to improve the result from a level to another one. The proposed solution combines join idle queue and join shortest queue approach.

In [21], a two level priorities are defined for tasks and resources. Using Min-Min, scheduler start by schedule the tasks of high priority to the high level resource, then schedule the rest of tasks to all available resources. Then, a rescheduling is made to improve the load balancing between resources.

In [92], a scheduling algorithm based on non linear programming is proposed while only considering the network bandwidth. According to the author, a smart usage of network bandwidth allows to avoid wastage of available resources.

In [80], an intelligent technique was proposed, by choosing dynamically in each iteration the adequate solution (RASA, Min-Min) to be used.

In [78], Carbon and Cost Aware GEographical Job Scheduling (CAGE) is a technique based on the Alternating Direction Method of Multipliers (ADMM) and envisage to reduce carbon footprints and electricity cost in geo-distributed collocation data-centers. Then, this work proposes a dynamic distribution of the work-flow considering local renewable availability, carbon efficiency, electricity price, and energy usage.

In [43], a dynamic preemptive resources allocation approach is proposed, the preemption is down to allow the tasks of high priority to be executed on concerned resource. The priority of tasks are based on cost and deadline. The proposed approach support also the fault tolerance.

In [46] a hybrid integrated thermal aware scheduling algorithm was proposed, which aims to minimize cooling energy consumption in data center labs when assigning jobs for computation.

In [105], Longest Cloudlet Fastest Processing Element (LCFP) and Shortest Cloudlet Fastest Processing Element (SCFP) are two algorithms which are proposed, both LCFP and SCEP envisages to optimize the total makespan. In *LCEP*, the length-

ier cloudlets are mapped to processing elements having high computational power, while *SCFP* starts firstly by mapping the shorter cloudlets to processing elements having high computational power.

In [20], a scheduling algorithm for assigning jobs to machines with heterogeneous processor cardinality is exposed. The authors have proposed to improve the efficiency of the dynamic programming, and have considered the total makespan as objective to be optimized.

In [113], the authors have proposed a scheduling heuristics that aims to reduce power consumption of parallel tasks in a cluster with the Dynamic Voltage Frequency Scaling (DVFS) technique. Then, the authors have discussed the relationship between energy consumption and task execution time.

In [98], a cost based algorithm was proposed, when a group of tasks are created according to their proprieties, then divide each group into sub-groups based on the resources' Million Instructions Per Second (MIPS) and the required Millions Instructions (MI) of each task. Finlay, applying the the proposed algorithm while minimizing the total cost.

In [94], an interesting point of the proposed Cloud workflow scheduling algorithm (CWSA) is its implementation in a real world by considering a scientific complex workflow applications.

2.2.2 Evolutionary Approaches

2.2.2.1 Conventional Evolutionary Approaches

Genetic Algorithm (GA) [42] is a popular meta-heuristic, considered as an evolutionary approach works in polynomial time, *GA* is inspired by the biologic process of the natural selection, in a standard *GA*, the algorithm starts with an initial population, where this latter is composed of a set of solutions called individuals (chromosomes), the initial population known as first generation as inspired from the biologic language, the *GA* looks to improve the initial population by applying two main operators $Crossover$ and $Mutation$, the new population called new generation. The passage from a generation to another is called iteration, $Crossover$ operator consists to exchange a parts (gens) of two selected individuals, while $Mutation$ operator consists to alter one or more gens with a given probability called crossover probability, usually the value of the latter is low.

A fitness function is used in the selection processes, by giving to best individual a high probability to continue its existence in the next generation, while the individuals

with lower fitness values will have a high probability to be dropped out. Then, a stop criterion is used to make an end to the algorithm, that can be a fixed number of generations, non evolution in the result after a number of generation, or a fixed time of execution of the algorithm.

Another nature-inspired meta-heuristic called Ant Colony Optimization (ACO) was proposed in [26], ACO has been inspired by the foraging behavior of ants, where a set of ants looks to find the best way to a found source of food. The algorithm works as follows. The ants start by searching randomly a way that can reach the food, when an ant finds a food, it comeback to the nest and deposits in the return way a chemical pheromone, and each other ant explores this way deposits a pheromone, the quantity of the deposed pheromone in a way allows to evaluate the quality of the way. Consequently, the way that contains the biggest quantity of pheromone is considered as the best know way. The vaporization of pheromone is a chemical property of the pheromone that is used, when a way contains a small quantity of pheromone is considered as bad way, then, it will be least used, subsequently, the pheromone will evaporate with time, until its disappearance, then the way disappearance. In an optimization problem, a way represents usually a solution, and the quantity of pheromone represents the quality of the solution. Several adaptations and modifications are proposed in the literature.

Particle Swarm Optimization (PSO) [103] is another meta-heuristic inspired from the living world, a set of particles work in collaboration in the swarm, each particle explores randomly the space of found object called solution, then saves its best found solution, each time a particle improves its best known solution, the latter will be communicated to the swarm in order to select a best global solution, finally, the best global solution is considerer a the final best found solution.

2.2.2.2 Non-conventional Evolutionary Approaches

In this subsection, we describe a set of algorithms that we have considered as non-conventional evolutionary approaches. That's means, the exposed algorithms start from an initial solution, and try to improve it to achieve a final solution. The algorithms are based on a conventional heuristics, conventional evolutionary approaches, or propose new operators.

Looking to the huge number of related work proposed in the literature, it is hard to describe all of them. Meanwhile, we have just selected a part of these works to be analyzed in this part of our thesis.

In the literature, many approaches base on GA are proposed, the authors of [97] and[39] are adapted GA for multi-objectives optimization.

Some other works combine GA with other heuristics in order to optimize one or more parameters, for example, Zhu et al. [134] have proposed a combination of GA with Multi-agent and Min-Min techniques, Shojafar et al. [104] have proposed a combination of GA with fuzzy theory, and Tao et al. [111] have combined Pareto and GA in the aim of optimizing the energy aware and the makespan.

Several propositions propose to improve the original GA by introducing simple modifications, like In [131] when the authors have considered computational and communication in their adaptation of GS, [38] when the authors propose to implement GA in an architecture based on Hadoop MapReduce using Java with GA package, [101] when the authors propose a shadow price guided SGA based approach, or [132] when the authors have proposed a parallel implementation of GA (PGA) to optimize the VM allocation in Cloud based environment. Also, in [91], a model was defined, then a multi-objective algorithm based on PSO was proposed.

Many approaches are proposed to make PSO more improved, [123], [44], [24] and [115] are good example that use PSO for multi-objectives optimization. The first one aims to optimize makespan and cost optimization, the second one looks to optimize the total processing cost and the communication time, the third one define load balancing, speed-up ratio and makespan as parameters to be optimized, then the last one envisages to optimize the cost and the task response time.

In [8], a few scheduling algorithms respect really the Cloud proprieties as elasticity and heterogeneities of resources was proposed, the proposed algorithm is based on PSO technique, a dynamic provisioning for the resources status is used in the proposed algorithm.

In [22], a set-based PSO (S-PSO) was proposed by addressing the QoS constraints. The S-PSO extends the original PSO in the discrete space and the service instances available in the Cloud can be considered as a resource set.The S-PSO was integrated with seven heuristics.

In [5], a variant of continuous PSO algorithm with smallest position value (SPV) and weighted sum approach for pareto-optimality was used to resolve bi-objective task scheduling problem in a Cloud Computing environment.

In [136], a resource allocation framework was proposed to execute a set of tasks on an external Clouds when the IAAS provider can not ensure the QoS requirement, the problem was formulated as an integer programming model, and solved using a self-adaptive learning particle swarm optimization (PSO) approach.

In [76], to ensure a better share of charge, the proposed approach ensure real time resource monitoring, and when a hot spots are identified, an ACO based algorithm is called, to redistribute the charge between resources. To evaluate the proposed approach, the authors have personalize an environment Jmeter and Xen.

In [25], an ant colony based algorithm was proposed to ensure a better load balancing, by exploiting the under loaded resources.

In [125], a tow level tasks scheduling algorithm was proposed, in the first level a PSO algorithm is applied, then in the second level an improved ants colony algorithm called max-min ant colony algorithm is applied.

In [72], a task scheduling policy using Load Balancing Ant Colony Optimization was proposed, by considering the MIPS and network bandwidth, the proposed algorithm was compared with basic ACO and FCFS. the compared parameters were makespan and the load balancing.

In [117], an ACO and PSO Combined algorithm was proposed and use the mutation and crossover operation inspired by the GA technique, according to the authors, the proposed algorithm accelerated the convergence speed and improve Resource Utilization Ratio.

2.3 Resource Management Solutions

A significant number of challenges have emerged with the advent of distributed computing technology. Consequently, an important number of works, which propose simulators and frameworks, were introduced in the literature in order to cope with these challenges. In the following, a selection of the most used and interesting ones is detailed. In this Section, we start first by presenting a set of simulators that can simulate such a distributed infrastructure, then we proceed to the presentation of a set of frameworks allowing to ensure the task of resource management in a real distributed computing infrastructures.

2.3.1 Simulators

SIMGRID [19] is a simulator that provides basic functionality for the simulation of distributed applications in heterogeneous distributed environments. Its use is well suited for the evaluation of heuristics, prototyping, development, and improvement of grid applications.

DCWorms (Data Center Workload and Resource Management Simulator) [70], is a simulator developed at the Poznan Supercomputing and Networking Center (PSNC)

laboratory in Poznan, Poland, based on the former GSSIM [3] simulator. GSSIM has been developed to provide an automated tool for experimental studies of resource allocation and scheduling policies dedicated to distributed systems. DCworms is developed on these basic GSSIM functionalities, but above all it offers additional functionalities related to energy consumption and energy efficiency studies. DCWorms has as its main objective to enable the modeling and simulation of computing infrastructures in order to estimate their performance, energy consumption, as well as different energy consumption metrics for different workload types and scheduling policies. The DCWorms simulator input data can be directly provided by the user, or generated using a GSSIM functionality, called workload generator. However, the key elements of DCWorms' architecture are plugins. They allow users to configure and adapt their simulation environment according to the characteristics of their studies.

GreenCloud [65] is an extension of the network simulator ns2 [52]. It provides a detailed modeling and analysis of the energy consumed by the elements of the network servers, routers and links between them. In addition, it analyzes the load distribution through the network, as well as communications with high accuracy (TCP packet level). In terms of energy, GreenCloud defines three types of energy: calculation (CPU), communications, and physical computing center (cooling system), and includes two methods of energy reduction: DVS (Dynamic Voltage Scaling) to decrease the voltage of switches and DNS (Dynamic Network Shutdown) that allows to shut down switches when it is possible.

CloudSim [16] is a toolkit for modeling and simulation of Infrastructure as a Service (IaaS) cloud computing environments. It allows users to define the characteristics of data centers, including number and characteristics of hosts, available storage, network topology, and patterns of data centers' usage. It allows the development of policies for placement of virtual machines on hosts, allocation of cores to virtual machines, and sharing of processing times among applications running on a single virtual machine. The application layer is managed by brokers, which represent users of the cloud infrastructure, that request creation of virtual machines in the data center. A broker can simultaneously own one or more virtual machines, which execute application tasks. Virtual machines are kept operating while there are tasks to be executed, and they are explicitly deallocated by the broker when all the tasks finish. Capacities of both virtual machines and hosts' CPUs are defined in MIPS (Million Instructions Per Second). Tasks, called cloudlets, are assigned to virtual machines and defined as the number of instructions required for their completion. Recently, a Fog version is available called iFogSim [45].. The latter is designed for modeling and simulation of

resource management techniques in the Internet of Things, Edge and Fog computing environments.

Many other open-source simulators based on CloudSim have been proposed, like NetworkCloudSim [36], FederatedCloudSim [67], DynamicCloudSim [13], TeachCloud [56], FTCloudSim [133], WorkflowSim [23] ElasticSim [15], CloudAnalyst [119] and CloudReports [96]. All cited simulators were developed using the Java language and include the support for energy, cost and federation models. Only Techcloud includes the SLA (Service-Level Agreement) support. Some of the simulators (ElasticSim, CloudAnalyst, CloudReports and TechCloud) provide a Graphical User Interface (GUI). A common disadvantage of those simulators is that they do not support parallel experiments, which means that they do not offer the ability to combine more than one machine to work together in order to process the given tasks.

ContainerCloudSim [90] and EMUSIM [17] are two important extensions of CloudSim. The first one allows to estimate the data center power consumption by defining a power model. While the second one proposes to integrate emulation and simulation, it combines emulation (AEF-Automated Emulation Framework) and simulation (CloudSim). In the emulation part, the software model is tested on the actual hardware itself.

Another proposed tool called EMUSIM [17] integrates emulation and simulation. It combines emulation (AEF-Automated Emulation Framework) and simulation (CloudSim). In the emulation part, the software model is tested on the actual hardware itself. EMUSIM is an open-source solution developed using the Java language.

An extension of the NS2 network simulator called GreenCloud [66] was proposed. It is an open-source tool developed using the C++ and OTcl programming languages and has a GUI. The users must use both C++ and OTcl to carry-on their simulations .

A commercial solution similar to CloudSim called MDCSim [74] was proposed. It is an event-driven simulator, developed using Java and C++ programming languages. MDCSim provides a graphical user interface that allows to easily model the desired Cloud architecture.

A flexible simulator called iCanCloud [86] based on SIMCAN [85] was proposed by Núñez et al.. The latter is a simulation platform designed for modeling HPC architectures and the former comes to avoid the common weakness other simulators such as CloudSim, GreenCloud and MDCsim. Both SIMCAN and iCanCloud are open-source simulators developed using the C++ language. Those two simulators

offer the possibility of making parallel experiments. With iCanCloud it is possible to add many adapted MPI library and POSIX based API for simulating new applications. However, none of the two simulators provides the energy model, the Federation model and, the SLA support.

secCloudSim is a simulator built on top of iCanCloud and proposed in [93]. It is developed using the C++ language and provides security basic features such as authentication and authorization. Similarly to iCanCloud, secCloudSim supports parallel experiments.

An open-source test-bed framework called OpenStackEmu [48] was proposed by Hernandez Benet et al., for experimental research on data communication networks, especially in the area of cloud infrastructures. According to authors, OpenStackEmu is the first attempt that combines the OpenStack infrastructure with a Software Defined Networking (SDN) based controller. OpenStackEmu enables emulating a large-scale network connected to the OpenStack infrastructure.

2.3.2 Frameworks

In [32], the authors have proposed and implemented an Opensource framework called Snooze, which allows the distributed management of VMs in private Clouds. In addition to the functionality already exist in other Cloud management platforms (OpenStack, Eucalyptus, and OpenNebula), Snooze implements the dynamic VMs consolidation as one of its base features. Another difference consists to the implementation of hierarchical distributed resource management. The management hierarchy is composed of three layers: local controllers on each physical node, group managers for managing a set of local controllers, and a group leader dynamically selected from the set of group managers and performing global management tasks. The distributed structure ensures the fault-tolerance and self-healing management, by avoiding single points of failure and automatically selecting a new group leader if the current one fails. Snooze also integrates monitoring of the resource usage by VMs and hosts, which can be leveraged by VM consolidation policies. These policies are intended to be implemented at the level of group managers, and therefore can only be applied to subsets of hosts. This approach partially solves the problem of scalability of VM consolidation by the cost of losing the ability of optimizing the VM placement across all the nodes of the data center.

An approach called Sandpiper was proposed in [120], which aims to ensure the load balancing in virtualized data centers using VM live migration. The main objective

of the proposed system consists to avoid host overloads referred to as hot spots by detecting them and migrating overloaded VMs to less loaded hosts. The authors applied an application-agnostic approach, referred to as a black-box approach, in which the set of VMs are observed from outside, independent of the nature of the applications hosted on the VM. A hot spot is detected when the aggregate usage of a host's resources exceeds the specified threshold for k out of n last measurements, as well as for the next predicted value. Another proposed approach is gray-box, when a certain application-specific data are allowed to be collected. The VM placement is computed heuristically by placing the most loaded VMs to the least loaded host.

Entropy is an Opensource VM consolidation manager for homogeneous clusters developed by Hermenier et al.. Entropy was implemented on top of Xen and focused on two objectives: maintaining a configuration of the cluster, where all VMs allocate sufficient resources, and minimizing the number of active hosts. To optimize the VM placement, Entropy applies a two-phase approach. First, a constraint programming problem is solved to find an optimal VM placement, which minimizes the number of active hosts. Then, another optimization problem is solved, which consists to find a target cluster configuration with the minimal number of active hosts that also minimizes the total cost of reconfiguration, which is proportional to the cost of VM migrations.

An Opensource called OpenStack Neat was proposed in [6]. The framework based on the OpenStack platform, and oriented to distributed dynamic VM consolidation in Cloud data centers. The framework is designed and implemented as a transparent add-on to OpenStack, which means that OpenStack Neat can be used under OpenStack without modification of the original installation, and without specific configuration. The proposed framework focus on live VM migration approach. To avoid performance degradation VMs are migrated from overloaded servers, and to increase the resource utilization and decrease the amount of energy consumed, VMs are migrated from under-loaded servers to switched them off.

Teixeira et al. have proposed a combination framework of Mininet (a well–known SDN emulator) and POX (an Openflow controller written in python), in order to support the simulation of SDN features in cloud computing environments. Mininet is used to emulate network topologies and data traffic in a data center running an OpenFlow controller in POX. Thanks to the use of Mininet and POX, the proposed framework provides practical results and a ready-to-use software in a real SDN environment. The simulation tool, however, is lacking support for cloud-specific features such as defining heterogeneous VM types or executing various application workloads

on the simulated host.

A Virtual Resource Management Framework (VRMF) is proposed in [84]. VRMF allows to manage the resources in IaaS Cloud federation. System performance evaluation is done using ns3 environment, while basing on response time to evaluate the ecosystem, and the system must be evaluated before and after the deployment of all modules in VRMF.

A general cross-layer Cloud scheduling framework for IoT tasks is proposed in [121], where an appropriate algorithm will be selected dynamically based on some criteria of the analyzed tasks, different experiments were conducted using CloudSim simulator.

In [135], a resource provisioning framework oriented QoS requirement was proposed ensuring an effective scheduling objective. The framework was evaluated by both simulation using CLoudSim and in real cloud environment based on aneka, on which energy consumption, execution cost, and execution time are the considered parameters.

Somasundaram and Govindarajan have proposed a solution called CLOUDRB. The latter is a cloud resource broker for scheduling and managing High-Performance Computing (HPC) applications in Science Cloud, it integrates deadline-based job scheduling policy with particle swarm optimization-based resource scheduling mechanism to minimize both cost and execution time to meet a user specified deadline.

CometCloud [63] is a cloud framework that provides autonomic workflow management by addressing changing computational and QoS requirements. The overarching goal of CometCloud is to realize virtual computational cloud infrastructure that integrates local computational environments and public cloud services on-demand, and provide abstractions and mechanisms to support a range of programming paradigms and real-world applications on such an infrastructure. Specifically,

2.4 Analyses

Reference	Objectives	Based on	Evaluation tool	Environment-oriented
[4]	Makespan	Min-Min	GridSim	Grid
[28]	Makespan	Max-Min	JAVA 6	Cloud Computing
[81]	Makespan	[88] and Min-Mi	Java 7	Cloud
[88]	Makespan	Min-Min, Max-Min	GridSim	Grid
[68]	Makespan, cost, balance load	Max-Min	CloudSim	
[73]	Makespan	Max-Min	MATLAB.	Grid
[116]	Load Balancing	Max–Min, machine learning	not cited	Mobile Cloud Computing
[87]	Load balancing and makespan	Min-Min, Max-Min	MATLAB	Grid computing
[60]	Makespan	Min-Min	Java	Cloud, and Grid computing
[7]	Makespan	[28]	CloudSim	Cloud computing
[50]	Completion time	Max-Min	CloudSim	Cloud Computing
[40]	Load balancing	Max-Min	Alea 3.0	Grid
[31]	Makespan	Min-Min and Max-Min	GridSim	Grid Computing

Table 2.1: Scheduling algorithms' comparison

Table 2.1 shows a brief comparison of the algorithms described above in terms of the objectives to be optimized, the used heuristics, the tools used in the evaluation phase, and the target environments. In the table, we are interested by a subset of the presented scheduling algorithms, which are the ones based on Min-Min, or Max-Min. The choice is done because one of our proposed heuristics uses Max-Min and Min-Min. The table shows that all presented algorithms, including all the other algorithms discussed in section 2.2, are evaluated by simulation, That's means, none of them has been deployed on a real distributed architecture. The lack of deployment of scheduling algorithms in a real-world environment can be explained by:

- The lacking of efficient solutions that facilitate the switch from simulation to real deployment,
- even if such a framework that allows a real-world deployment exists, it is difficult to make an easy switch from one architecture to another.

As described in Section 2.3, the already known resource management solutions, including frameworks and simulators, propose different kinds of relevant features. None of them includes an easy solution to switch between different environments. This leads us to suggest the need to propose a framework, which eases the switch between a theoretical study, an emulator, and a real distributed platform. The switch between different kinds of environments may be done automatically, without additional efforts. Indeed, a theoretical, or local emulated study, is often relevant when starting a research study as it allows to have first results without requiring a long preparation time.

To our best knowledge, all existing frameworks are designed for a specific architecture, which is considered as common weakness between these frameworks. This common weakness can be resolved by proposing a solution that facilities the switch from one architecture to another, for example from a Cloud based on Openstack to a Cloud based on Azure, from a Cloud based on OpenStack to an emulated environment based on Emulab, ...etc. The issue is addressed in our thesis work, on which we have proposed a framework that contributes to resolving this weakness. Our proposed framework is described in detail in Chapter II.

2.5 Conclusion

In this chapter, we first presented a set of scheduling algorithms in relation to our propositions. Then, in the second time, we have presented a set of evaluation tools in relation to resource management, especially the ones used in the field of distributed systems.

In the first part, we started by presenting a set of simple scheduling heuristics separated into two sub-sections. In the first one, a set of convention heuristics, then a set of heuristics based on both Max-Min and Min-Min that is in relation with our proposition MMin are presented. Then, in the second one, we have discussed a set of conventional evolutionary approaches, followed by the presentation of a set of non conventional evolutionary approaches

In the second part, we have presented a set of evaluation tools, that can be used as a resource management solution. The presentation was separated into two parts. In the first one consist of showing a set of simulator, that can simulate distributed computing infrastructures, while the second part consists of presenting a set of framework able to manage a resource of an infrastructure. the frameworks are, usually, used for a specific case. Unfortunately, all the presented evaluation tools can't make the priority of "From theoretical to real deployment. The latter is the subject of one of our contributions, which will be presented in detail in the following chapter.

CHAPTER III

MFHS: a Modular scheduling Framework for Heterogeneous System

3.1 Introduction

Current innovative distributed architectures, proposing on-line services, involve more and more computing resources. From a provider point of view, the platform management leads to challenging problematic relating to resource allocation which involve different kind of quality of service parameters the provider has to focus on to keep his platform reliable and efficient. In this chapter, we will present in detail our proposed tool called *MFHS*, which is a modular generic framework that can be adapted to any distributed computing environment. Structured in modules. *MFHS* allows to discover the existing computing resources in terms of computing performance, network throughput and disk I/O speeds (Resources Discovery module), it also allows to predict how the experiment should behave (P_i value). As the setting up of real experiments is often complex, *MFHS* allows: to make theoretical experimentation (based on models), to use any kind of distributed emulators or to deploy experiments on real experimental platforms. Our framework take into consideration four QoS parameters (Resources Monitoring module): energy consumption, cost, resource utilization, and makespan. Then, the users of our framework can explore the heuristics already deployed on *MFHS*, as can deploy any new task scheduling heuristic.

In more practical terms *MFHS* allows to conduct studies in both theoretical and real environment. In other words, *MFHS* can be used as a simulator only to conduct theoretical studies using generic input before deploying it on a real distributed computing environment. This kind of pre-computation phase can help to conduct a smart first evaluation of any scheduling approach. On the experimental side, *MFHS*

framework can be deployed on any distributed computing environment. Its behavior and architecture allow to discover the environment on which *MFHS* is deployed by checking various properties, such as configuration files, network characteristics, hardware performance, etc.

From an algorithmic point of view, *MFHS* can support any scheduling algorithm taking into account single or multi-objective QoS approach. In the proposed architecture, the *Scheduling* module is in charge of executing the chosen algorithm to find an affectation of a set of tasks to a set of resources. A task is the basic element which is characterized by a set of needs in terms of the amount of data to upload, download, write and read from disk. Also, the technique used for the invocation of the scheduling process is the time invocation. The task characteristics supported by the *MFHS* are independents, non preemptive, with or without deadline and without priority.

In the reminder of this Chapter, we start by describing the global architecture chosen for *MFHS* in Subsection 3.2. Then, the whole modules are introduced separately in Subsection 3.3. Finlay, Subsection 3.4 shows in detail several QoS parameters used in the following studies.

3.2 High level Architecture

Figure 3.1 depicts a high level description of the considered computing architecture. The set of resources are hosted on different datacenters linked by a network which could be in distinct geographical location. Each of these datacenters hosts a set of computing resources (also called computing nodes), which can be physical servers, as can be virtual machines. Each computing node has its own characteristics resulting to the heterogeneity of the distributed computing resources. The characteristics that make resources heterogeneous can be pure processor performance, network bandwidth, I/O speed, etc.

Figure 3.1 depicts the global architecture proposed from the users where which come from the requested task (T_1,...,T_n) to the computing nodes (VM_1,...,VM_m) that are the parts of the available physical resources (PM).

3.3 Internal architecture of *MFHS*

Figure 3.2 describes in detail the internal architecture of *MFHS* framework.

We have chosen a modular architecture, in order to enjoy the benefits of modular

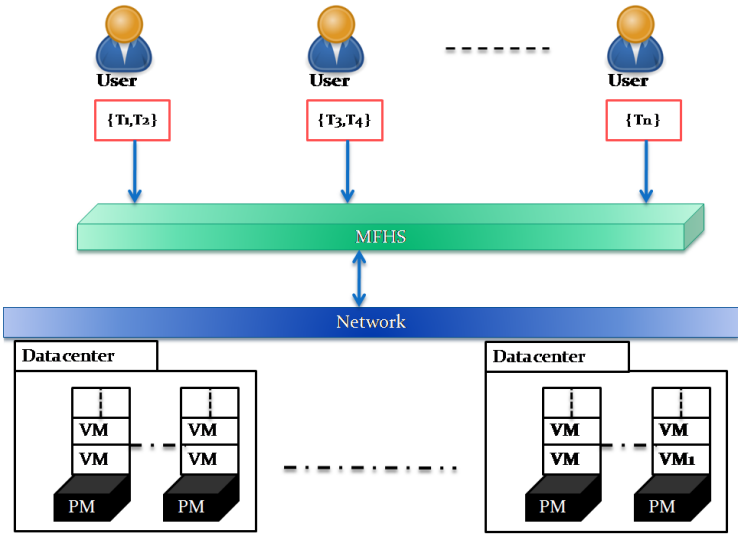


Figure 3.1: High level global architecture

development, such as facilitating the understanding of the source code to make fast modifications and adaptations for any distributed architecture. It is also possible to take only a subset of the modules for re-utilization in another resource management solution. Each module of *MFHS* framework serves to fulfill a well defined function.

MFHS is a centralized framework, meaning, to deploy the *MFHS* on a new or an existing distributed computing architecture, the framework can be deployed on any chosen simple machine.

From a security point of view, all exchanged messages done during MFHS processes are encrypted, which avoid any risk of learning messages if the latter are sniffed. Two possibilities are possible, use asymmetric cryptography or symmetric cryptography. The first one, also known as public-key cryptography, is a process that uses a pair of related keys, one public key and one private key, to encrypt and decrypt a message and protect it from unauthorized access or use. A public key is a cryptographic key that can be used by any person to encrypt a message so that it can only be deciphered by the intended recipient with their private key. While the second one, is a method where the uses a single key that needs to be shared among the people who need to receive the message.

In large distributed systems, including our case, the use of symmetric cryptography, implies a huge number of keys that must be exchanged over the network, but in the case of using asymmetric cryptography, this number of exchanges is reduced. For this reason, we have opted for the asymmetric method. Then, the Secure Shell (SSH)

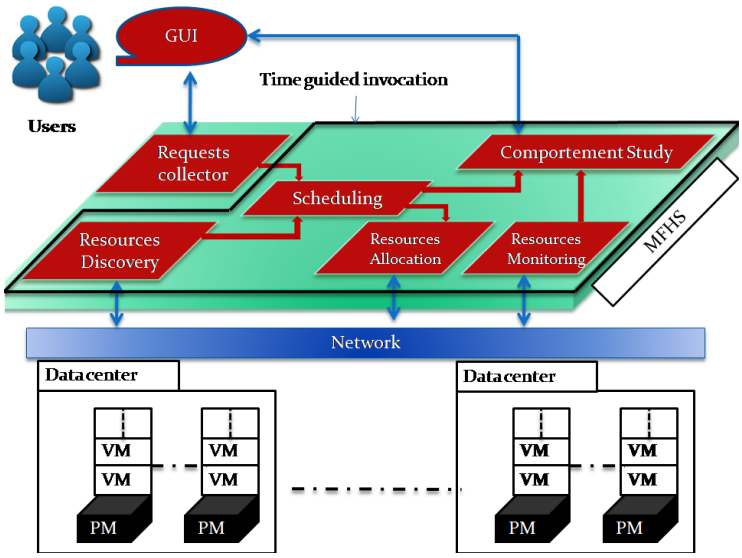


Figure 3.2: Internal components of the scheduler

protocol with a public-key authentication is used to ensure a secure communication between the *MFHS* framework and the set of computing nodes, and any change is needed at the level of the computing nodes.

In the following, We will give the details of each module of the *MFHS* framework.

3.3.1 Resources Discovery

This module makes carry out the Resource Discovery phase. This data collected by this module will be used by the *Scheduling* module and the *Resources Monitoring* module so the different informations that are collected by this module will be stored in a structured file. Where each line corresponds to the capacity description of one resource (j), identified by a unique identifier R_j, each line of this file is written using the following structure:

Resource description

$$RPD_j = R_j : PM_j : DebitDk_j : DebitUk_j : SpeedI_j : SpeedO_j : NBRPhysicalCPUs_j : NBRLogicalCPUs_j : P_{min_j} : P_{max_j} : CR_j : SMem_j : SDisk_j$$

In the previous structure (Resource description), RPD_j represents the performance description of the resource R_j, PM_j represents the identifier of the physical machine that hosts R_j, $DebitDk_j$ represents the download speed from the resource

R_j, $DebitUk_j$ represents the upload speed on the resource R_j, $SpeedI_j$ represents the write rate on the disk of the resource R_j, $SpeedO_j$ represents the read rate from the disk of the resource R_j, $NBRPhysicalCPUs_j$ represents the number of the physical CPUs available on the resource R_j, $NBRLogicalCPUs_j$ represents the number of logical CPUs available on the resource, P_{min_j} represents the power consumed at the idle state of the PM_j processors, P_{max_j} represents the power consumed at the 100% use of the PM_j processors, CR_j represents the utilization cost of the resource R_j per unit time, $SMem_j$ represents the memory size of the resource R_j, and $SDiks_j$ represents the storage disk size of the resource R_j.

The set of values are retrieved directly if the environment on which *MFHS* is deployed makes it possible to set the values concerning upload and download rates, read and write rates on the disk. But some cloud environments do not give the possibility to fix these values, in this case, this module allows to discover the different average debit, by executing a set of scripts.

In addition, this module allows getting a view about the environment stability on which *MFHS* is deployed, and this by computing a convergence index called Pi, whenever Pi close to 1 then the environment is more stable, in order Pi is calculated using Formula **??**, 5.3 and 5.5. This set of three formulas are defined and described in Subsection 5.3.6.2. Knowing Pi the designer of a scheduling algorithm can predict the degree of precision of the results in terms of QoS to be obtained before running the algorithm, and to predict the possible difference between theoretical and real results.

3.3.2 Requests Collector

This module collects data about the tasks execution requests, information collected during the execution of the *Requests Collector* module are stored in an single structured file. Each line describes a unique task, the structure of this file is as follows:

Request description

$$PRTi = T_i : SizeDTU_i : SizeDTD_i : SizeDTW_i : SizeDTR_i : TimeCPU_i : EstimatedCPU_i : RNBRPhysicalCPUs_i : RNBRLogicalCPUs_i : RMem_i : Budget_i : MaxEnergy_i$$

In the previous structure (Request description), $SizeDTU_i$ represents the size of data to be downloaded by the task T_i, $SizeDTD_i$ represents the size of data to be uploaded by the task T_i, $SizeDTW_i$ represents the size of data to be writing in the disk during the execution of task T_i, $SizeDTR_i$ represents the size of data to be reading

from the disk during the execution of task T_i, $TimeCPU_i$ represents the time on which CPU will be used to $EstimatedCPU_i\%$ of its capacity, $RNBRPhysicalCPUs_i$ represents the number of physical CPUs required by the task T_i, $RNBRLogicalCPUs_i$ represents the number of logical CPUs required by the task T_i, $RMem_i$ represents the memory size required by the task i, $Budget_i$ represents the budget of the task T_i that must not be exceeded, and $MaxEnergy_i$ represents the maximum energy that a task T_i must not be exceeded.

The information mentioned in the corresponding file makes it possible to filter the whole of the available resources in order to select only the resources that owns the necessary performances in order to candidate to the execution of the concerned tasks. Next, these values will be used to calculate the estimated value of execution time, cost, and consumed energy of each task on each resource.

3.3.3 Scheduling

Scheduling module begins by collecting the information gathered by the *Resources Discovery* module and the *Requests Collector* module. Then, two sub steps are applied. In the first one step and for each task T_i, a filter on the set of available resources is applied, in the objective of extracting only the computing nodes that be able to execute the given T_i, this filter is realized in function of the compute node capacity on the one hand, and other in function of the needs of the task. Let us point out that the parameters supported in the filter are: minimum needed memory, minimum needed storage space and the number of the required CPUs and vCPUs. In the second step, the set of selected schedulers are applied.

Each scheduler must have in input, the set of tasks, the set computing nodes, the matrix that estimates the execution time of each task on each available resource, matrix that estimates the cost of the execution of each task on each resource, and the matrix that estimates the energy consumed by each task in function of the resource. Then, each called scheduler, must generate as output a matrix, that is called an allocation matrix, this matrix is of size $2 * n$, where n is the total number of tasks that must be executed, the first column represents the task identification and the second column represents the computing node identification that is to execute the given task.

The *Scheduling* module is modeled in such a way that it is easy to inject any kind of scheduling algorithm. Each integrated algorithm must return an affectation matrix, and can have one ore more matrix as input.

The different steps of this scheduling module are listed the following Algorithm 1:

INPUT Tasks description
Resources description
FUNCTION Apply the filter
Compute estimated execution time (Equation 3.3) of each task on each resource
Compute estimated cost (Equation 3.7) of each task on each resource
Compute estimated energy consumption (Equation 3.10) of each task on each resource
Call the appropriate scheduling algorithm
OUTPUT Assignment of tasks to resources

Algorithm 1: Scheduling Module Algorithm

To compute the estimated execution time of each task on each resource the scheduler uses (Equation 3.3, while the estimated cost depends to the estimated execution time ((Equation 3.7)). The energy consumption can be estimated by knowing the CPU use ((Equation 3.10)).

3.3.4 Resources Allocation

The main role of this module is to perform a resources allocation based on an affectation matrix. Next, this module is also responsible for sending each task to its corresponding computing node. For each task, the source code and the data must be uploaded to the corresponding computing node. At the end of the task execution, the resulting data must be downloaded from the corresponding computing node.

This module must record the start and end times of the task execution. This information will be used by the *Behavior Study* and *Monitoring* modules.

3.3.5 Resources Monitoring

The resources monitoring module is a system process, created at the beginning of the global task scheduling process invocation. This module is useful when it is a real execution, but when *MFHS* is called for a theoretical analysis the process corresponding to this module will not be instantiated. This process stays active during the execution of all other modules of *MFHS*. Since all modules have finished their execution, the resource monitoring process is killed and waits to be reactivated

for the next scheduling processes. This module allows controlling the platform on which *MFHS* is deployed during the real execution of the task scheduling process. It can detect any anomaly such as the loss of connection to a compute node, erroneous access attempts that may be the object of an attack or the loss of connection on the node where *MFHS* is running. The various information extracted from this process is archived and will be accessible for future uses.

3.3.6 Behaviour Study

This module is invoked at the end of the scheduling process. It allows to collect informations from all other module and generates a detailed report containing, for each algorithm, the following real and theoretical values:

- The total response time of each compute node, the average response time and the makespan,
- The compute nodes resource utilization and the overall average resources utilization,
- The energy consumed by each resource and the overall average energy consumption,
- The total cost,
- The data transfer (upload and download) rates which is the average between the control node and the computing nodes,
- The read/write rates average on disk.

Having these theoretical and real values, the module computes a ratio using these values in order to estimate the sustainability of the computing environment. In addition, the report contains a set of information about the possible encountered difficulties during the execution of scheduling process.

3.4 MFHS QoS measurement

In the following, formulas and descriptions of the QoS parameters which are taken into account in *MFHS* in terms of completion time, resource utilization, cost and energy consumption are described.

3.4.1 Total response time (makespan)

The makespan, MS, represents the maximum over the tasks completion times. It is computed using formula 3.1, where $NbrT$ represents the total number of tasks.

$$MS = \max\{(CT_i)\ ,\ i = [1...NbrT]\} \tag{3.1}$$

3.4.2 Response time

Total response time (makespan), MS, is computed for the whole of all tasks, and it is computed when the execution of all the tasks is finished. Meanwhile, the completion time (CT_i) is computed for each task T_i, using formula 3.2. In the latter, $TStart_i$ represents the start time of the task T_i, and ET_i represents the execution time of the task T_i on the resource R_j. ET_i is obtained using the formula 3.3.

$$CT_i = TStart_i + ET_i \tag{3.2}$$

$$ET_i = \frac{SizeDTW_i}{DebitI_j} + \frac{SizeDTR_i}{DebitO_j} + \frac{SizeDTD_i}{DebitDk_j} + \frac{SizeDTU_i}{DebitUk_j} + TimeCPU_i \tag{3.3}$$

3.4.3 Average resources utilization

The optimization of the resources usage is one of the objectives of QoS, this function must be maximized, it is computed by the following formula:

$$RU = \frac{\sum_{j=1}^{m}(RU_j)}{n} \tag{3.4}$$

In Formula 3.4, RU represents the resource utilization average of the whole set of all resources, and RU_j represents the resource utilization of R_j. RU_j is found using Formula 3.5.

$$RU_j = \frac{\sum_{i=1}^{n}(b_{ij} * ET_i)}{Makespan} \tag{3.5}$$

For each task T_i, the coefficient b_{ij} takes the value 1 if T_i is executed on R_j, else, the value b_{ij} equals 0.

The value of RU is between 0 and 1, each time the value is close to 1 then we say that the resources are better used

3.4.4 Cost

The total cost CU is the cost that a user must pay to service provider for resource utilization. This value may be minimized as much as possible, CU is computed using

the formulas 3.6 and 3.7.

$$CU = \sum_{j=1}^{m}(CU_j) \tag{3.6}$$

CU_j represents the cost of utilization of the resources R_j, and it is computed using the following formula:

$$CU_j = \sum_{i=1}^{n}(b_{ij} * ET_i * CR_j) \tag{3.7}$$

where CR_j denotes the cost of resource j per unit time.

3.4.5 Energy consumption

According to the most important work on the energy consumption [95], the energy consumed by the server processors represents the most important values comparing with the other component of the server, like disk storage, network cards, etc. Formula 3.8 allows to get the power consumed by a processor, and subsequently by the server.

$$P = P_{min} + (P_{max} - P_{min})u \tag{3.8}$$

Where P_{min} represents the power consumed at the idle state of the processor. P_{max} represents the CPU power consumption when it is used at 100%. u represents the cpu load. Then, to get the power consumption of a resource j, the following formula is used:

$$P = (P_{max} - P_{min})Bj \tag{3.9}$$

Where Bj represents the Power consumed by the resource processes on the physical resource that hosts the resource i. Note that the power consumption of a resource j is captured n times during the execution of task i, then, to get the total energy (E_j)consumed by this task i on a resource j during T instances of time, the used formula is described as:

$$E_j = \frac{\sum_{t=1}^{n}(P_j)}{n} * T \tag{3.10}$$

The total energy consumed by the set of tasks is obtained using the following formula,

where $NbrR$ represent the total number of used resources.

$$E = \sum_{j=1}^{NbrR} (E_j) \tag{3.11}$$

3.5 Conclusion

As described above, distributed computing paradigm intrinsically allows numerous various kind of on-demand services which run on many computing resources. Thanks to these offers, Distributed computing infrastructure usage is still increasing and the QoS parameters can be widely affected with the size grown of the architectures in terms of the number of resources deployed and in terms of the number of users. The scalability motive to the exiting of a smart solution that can manage the architectures.

The importance of the task scheduling in a distributed computing architectures implies the existence of a large number of good propositions that look to solve the problematic, but almost of them have been evaluated by simulation and never been implemented despite their effectiveness, this due to the difficulty of moving from simulation to real implementation, and even if it has been implemented, it is also difficult to switch from one platform to another. In this chapter, we have presented our solution that can contribute to solving these issues.

From a service provider point of view, one of the main keys is the management of the computing resources which continuously receive a very large number of requests. The management must consider the optimization of QoS parameters. In our presented contribution, a set of QoS parameters are considered and analyzed. For example, the energy consumption that is now well known and raise much research problematic and which has an environmental impact, the cost of services consumption which has an important impact of a lot of companies, and the response time and load balancing can not be ignored which has an indirect impact on the energy consumption and cost.

In our best knowledge, the already known frameworks propose different kind of relevant features. None of them includes an easy solution to switch between different environments. While introducing *MFHS*, we propose a framework which ease the transition between a theoretical study, an emulator, and a real distributed platform without hard efforts. Indeed, a theoretical, or local emulated study, is often relevant when starting a research study as it allows to have first results without requiring a long preparation time.

CHAPTER IV

Scheduling Algorithms

4.1 Introduction

Independent task scheduling problem in distributed computing environments with *makespan* optimization as an objective is an NP-Hard problem. Consequently, an important number of approaches looking to approximate the optimal *makespan* in reasonable time have been proposed in the literature. In this chapter, we present two new heuristics of independent task scheduling, the first one called *MMin* and the second one called *InterRC*.

InterRC solution is an evolutionary approach, which starts with an initial solution, then executes a set of iterations, for the purpose of improving the initial solution and close the optimal *makespan* as soon as possible. During the *makespan* improvement some operators are used, which will be described in detail in this chapter.

MMin is a fast deterministic heuristic inspired by *MinMin* and *MaxMin*. Indeed, *MinMin* and *MaxMin* both have advantages and disadvantages regarding the tasks and resources characteristics. Our proposed *MMin* algorithm makes a hybridization of *MinMin* and *MaxMin* in order to take advantage of these two algorithms. The main objective of *MMin* consists in optimizing the total *makespan* while preserving the fast execution of the algorithm that characterize both *MinMin* and *MaxMin*.

4.2 Target Environment

In both, this chapter and the next one, we present our contributions. In all our contributions, we look to be independent on the used environment. Thus, both MMin and InterRC algorithms, that we present in this chapter, are intended for any distributed algorithms. Meanwhile, whatever the level of resources' heterogeneity ,

and of tasks' heterogeneity our algorithms can be used. Grid, Cloud, Fog computing, and other systems that we have described in Chapter I are good examples of systems, on which our algorithms can be deployed.

Figure 4.1 shows a simple task scheduling scheme, which represents a kind of MMin, and InterRC deployment. The set of resources that compose the distributed architecture to be managed can be situated at any geographical localization, with condition that any two distinct resources can be connected with direct or indirect link. Then, concerning the tasks, we assume that a given task can come from any foreign resource, or be located on a resource owned by the distributed system that we want to manage.

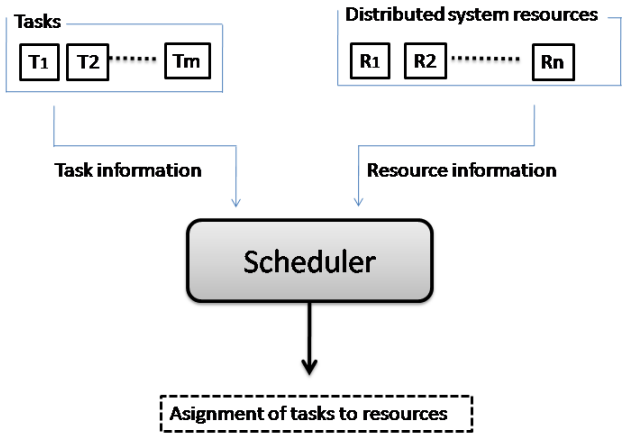


Figure 4.1: Task scheduling scheme

In the next section, we describe the formulation of the task scheduling problem that we have considered in this thesis.

4.3 Problem Description

The task scheduling problem addressed in our thesis can be described as follows: a set of tasks $T = \{T_0, ..., T_{n-1}\}$ must be mapped to a set of resources $R = \{R_0, ..., R_{m-1}\}$, where n represents the total number of tasks to be scheduled and m represents the total number of available resources allocated to execute the whole set of tasks T. Assuming that a matrix ET of $n * m$ elements is defined, where each

entry $ET[i][j]$ of the matrix ET represents the execution time of the task T_i on the resource R_j whatever $T_i \in T$ *and* $R_j \in R$.

It is assumed that all tasks to be scheduled are independent and non-preemptive, i.e. There is no dependency relationship between the tasks and whatever T_i, T_i cannot stop once started. In addition to these two conditions (independence and non-preemptively), all tasks have the same priority, that is, for each couple (T_i, T_j) in T, T_i and T_j have the same chance to be scheduled first.

The model used to estimate the execution time of each task T_i in T on each resource R_j in R is the one defined by Ali et al.[2]. This model is considered as one of the most widely used models for Heterogeneous Computing Scheduling Problem (HCSP). When generating the matrix ET this model takes into account three properties: machine heterogeneity, task heterogeneity, and consistency.

Machine heterogeneity represents the relation between resources in terms of computing power, which results in a variation of execution time. In a high machine heterogeneity HCSP systems, the difference between the execution time of a task T_i from a resource to another is high. On the other hand, in low machine heterogeneity, the difference between execution times is low. The task heterogeneity represents the difference of computing power needs from a task to another. Subsequently, in a high task heterogeneity HCSP, there is a high difference in terms of execution time from a task to another on a given resource R_j. In contrast, for a low task heterogeneity HCSP this difference remains low. The third classification used in HCSP is the consistency: in a consistent ETC, if a task T_i is slower than a task T_j on R_j, then T_i is slower than a T_j on all other machines. An ETC is inconsistent if it is possible to find two tasks T_i and T_j such that T_i is slower than T_j on some machines and T_j is slower than T_i on other machines. Moreover, a semi-consistent ETC can be used to model those inconsistent systems that include a consistent subsystem.

4.4 MMin

An important motivation that led us to think about proposing a heuristic based on Min-Min and Max-Min is the complexity of the two algorithms. Both Min-Min and Max-Min are well-known heuristics and characterized by a low execution time (low complexity). Meanwhile, MMin is also characterized by its low complexity time, which is an important parameter that must be considered when treating a scheduling problems, especially when dealing with large scale distributed environments.

Both algorithms are efficient in terms of *makespan*, when the task's sizes are

quite the same in terms of execution time (low task/resource heterogeneity). However, when it comes to tasks with very different execution times (high task/resource heterogeneity), the both algorithms efficiency become to be more critical. In more detail, when there is a large execution time difference, the *Min-Min* algorithm penalizes the largest tasks leading to increase the completion time. Especially when there is a large amount of small-sized tasks [99]. For the *Max-Min* algorithm, it shows its advantages when there is a significant number of large tasks. In this case, the algorithm penalizes small tasks, by increasing their completion time mostly because they have to wait until all large tasks are finished [99].

Our proposed *MMin* heuristic strives at good performance for any level of task/resource heterogeneity, by trying to get to the situation where there are no large differences between the tasks' execution times. By doing this, the proposed method avoids the weaknesses of the two algorithms, *Max-Min* and *Min-Min*.

In the proposed heuristic, as described above, it assumed that we have a meta-task composed of n tasks, each task must be affected to a resource of the set of m available resources. *MMin* is executed in n (number of tasks) iterations. At each iteration, only one single task is scheduled and then deleted from the meta-task, then *MMin* algorithm ends when the meta-task becomes empty. The choice of the task to be scheduled is done with the objective of reducing the difference between the task's execution time (tasks not yet scheduled), i.e. making the system in "low task heterogeneity" and "low resource heterogeneity" stat.

4.4.1 Step 1: Task Classification

In this setup (marked Step 1 in Algorithm 2), the set of task are classified into categories according to their sizes, which allows us to get more informations about the global status of the system.

Two categories are defined, large tasks and small tasks, the first category contains the tasks with high execution time, and the second one contains the tasks with low execution time. The number of small tasks (GlobalNBRST) and the number of large tasks (GlobalNBRST) are varied from an iteration to another, therefore, are computed at each iteration. GlobalNBRST and GlobalNBRST are used in the second step.

The number of small and large tasks correspond to the number of tasks that have a completion time lower and greater than the *localavg* respectively. The *localavg* is computed for each resource for the set of tasks not yet scheduled. The *localavg* is computed, in the first two lines of the *forall* loop, using $max\_CT\_R_j$ and $min\_CT\_R_j$

```
while Exist tasks in meta-task do
    GlobalNBRST = 0;
    GlobalNBRLT = 0;
    Step 1:
    forall Resource R_j do
        range = max_CT_R_j − min_CT_R_j;
        valavg = min_CT + range/2;
        forall Tasks T_j do
            if CT(tj) < valavg then
                GlobalNBRST = GlobalNBRST + 1;
            else
                GlobalNBRLT = GlobalNBRLT + 1;
            end
        end
    end
    Step 2:
    if GlobalNBRST > GlobalNBRSLT then
        CallMax − Minalgorithm;
    else
        CallMin − Minalgorithm;
    end
    Delete scheduled task from meta-tasks;
    Update the completion time;
end
```

Algorithm 2: *MMin* Algorithm

which respectively represent the maximum value and the minimum value of the completion time of the resource R_j.

4.4.2 Step 2: Algorithm Selection

In this setup (marked Step 2 in Algorithm 2), one of the two *Max-Min* and *Min-Min* heuristics is called following few conditions. To decide which algorithm has to be executed, the number of the small tasks and the number of large tasks are compared. If the number of large tasks is higher than the small ones, the *Min-Min* algorithm is called. Otherwise, the *Max-Min* algorithm is called.

This second step allows to lead the system to stat of low task heterogeneity/ low resource heterogeneity, then, as explained previously in this section, avoid disadvantages of Min-Min and Max-Min heuristics.

4.5 InterRC Heuristic

In this section, we introduce the proposed heuristic, by first describing the details of the proposed *InterRC* heuristic, which aims to resolve the Heterogeneous Computing Scheduling Problem (HCSP) described previously in Section 4.3 of this chapter, The different elements needed for the understanding of *InterRC* heuristic are presented, as well as the process flow, will be presented in detail in five sub-sections.

4.5.1 Objective

The main objective of the *InterRC* heuristic consists in optimizing the *makespan* of HCSP described above, where *makespan* represents the total time needed to complete the execution of all tasks. The latter can be computed using Formula 4.1. In the formula, $ET[i][j]$ represents the execution time of T_i on R_j as already described in the previous subsection and b_{ij} is a boolean value which is equal to 1 if T_i is affected to R_j, otherwise b_{ij} equals to 0.

$$\begin{aligned} makespan &= max_{j=1...m} \sum_{i=0}^{n} ET[i][j] * b_{ij} \\ With, b_{ij} &= \begin{cases} 1 & if\ T_i\ executed\ on\ R_j \\ 0 & else \end{cases} \end{aligned} \tag{4.1}$$

4.5.2 Operators

The operators that we have defined allows to create new solutions, then give more chance to obtain an improved affectation of the set of tasks to the set of available resources.

The following two operators are defined, and used by the *InterRC* heuristic.

move: consists to test if the value of *makespan* obtained after a move of a task T_i from a resource R_{j1} to another one R_{j2} will not exceed the actual *makespan*. If that is the case, the move is applied, otherwise, it will not be applied.

permutation: consists in checking if the *makespan* after a permutation between two distinct tasks T_{i1} and T_{i2} situated on two different resources R_{j1} and R_{j2} will not exceed the actual *makespan*, then the permutation is applied in case the test proves to be true, if not, it is not applied.

4.5.3 End Conditions

Our *InterRC* heuristic comes to an end if one of the two following conditions is reached: The time $MaxResTime$ which represents the maximum time dedicated to the execution of *InterRC* algorithm is reached, or the loop $L1$ presented in both algorithms (Algorithm 5 and Algorithm 6) ends with any improvement of the actual *makespan* in the case of Algorithm 5, or with any possible redistribution in the case of Algorithm 6.

4.5.4 Global Process

As described in Algorithm 3, the global process of *InterRC* heuristic consists in three phases: The first one aims to generate an initial solution, while the second one looks to improve the actual *makespan*. Finally, the third phase allows to redistribute the set of tasks over the set of available resources without exceeding the actual value of *makespan*. Note that the second and third phases are executed alternatively with a fixed number of times ($NbrIterations$). The alternation allows a maximum redistribution, and avoids system stability as much as possible. The stability means the difficulty to find any moving/permutation that can improve the actual *makespan*. The second and third phases can be achieved using *permutation*/*move* operators.

Call Algorithm 4
i=1;
while $NotEndExecTime\ AND\ improve = true\ AND\ i \leq NbrIteration$ **do**
  i++;
  Call Algorithm 5;
  Call Algorithm 6;
end
Return Task affectation;

Algorithm 3: Global *InterRC* Process

4.5.5 Detailed Process

In this section, we aim to give more details about the *InterRC* heuristic process, by presenting the detail of the algorithms including in Algorithm 3, which is presented in the previous subsection.

A. Initial affectation: This first phase described by Algorithm 4 consists in generating an initial solution, which starts by choosing randomly a resource R_{init} and a task T_{init}. Then, the loop $L1$ of Algorithm 4 is called to assign all tasks

to R_{init} starting by T_{init}. Note that the choice of R_{init} and T_{init} has an impact on the final *makepan*, i.e. if we change the initial solution, we get probably a different final result.

Choose randomly a resources R_{j_init} AND a task T_{i_init}
for $(ii = 0; ii < n; ii++)$ **do**
  $i = (ii + i\_init)\%n$;
  Assign T_i to R_{j_init} ;
end
Return Task affectation;

Algorithm 4: Initial affectation

B. *makespan* improvement: In this second phase of the *InterRC* heuristic which is described by Algorithm 5, a loop over the set of tasks affected to R_{jMS} is executed (R_{jMS} represents the resource that gives the *makespan*). Then, for each found task T_{ims}, the algorithm loops over the set of other resources (other than R_{jMS}) and for each found resource R_j with $R_j! = R_{jMS}$, the algorithm first tests if the *move* of T_{ims} from R_{jMS} to R_j improves the actual *makespan*; if the test is positive, the *move* operation is realized, the new *makespan* is computed, and loop $L1$ broken with the update of *improve* value to true. Otherwise, the loop $L3$ are triggered with the aim of finding a task T_{i1} that can improve the actual *makespan* if it is permuted with T_{ims}, as soon as a task $Ti1$ is found, the *permutation* is realized, the new *makespan* is calculated, and the loop $L3$ is broken with the update of *improve* value to true.

C. Task redistribution: Algorithm 6 describes this third phase of the *InterRC* heuristic. Unlike Algorithm 5, Algorithm 6 will not try the *move* of a task from R_{jMS}, or the *permutation* of a task situated on R_{jMS} with another task located on another resource other than R_{jMS}. However, the tests will be larger. In more details, the loop $L1$ allows to loop over the set of resources R, then, for each found resource R_{j1}, a loop of the tasks affected to R_{j1} is done. Afterward, the algorithm re-loop over the set of resource and for each found resource R_{j2}. The algorithm test in the first time if the *move* of T_{i1} to R_{j2} will not penalize the actual value of *makespan*, then the *move* is applied in case the test proves to be true, if not, the loop $L3$ is triggered in order to find a task T_{i2} that does not penalize the actual *makespan* if it is permuted with T_{i1}. As soon as a permutation is possible, the *permutation* is realized, the new *makespan* is calculated, and the loop $L3$ is broken with the update of *improve* value to true.

In this third phase, each task can be moved/permuted only once. To ensure this uniqueness of *move/permutation*, a set called Ts_Tested is created, then, the tasks candidate to moving/permutation operators are added to this set. Thereafter, the tasks of the set Ts_Tested cannot re-participate to another *move/permutation* operations. The use of Ts_Tested set allow to finish the phase as fast as possible, with maximum *move/permutation* possible.

```
improve = true
L0: while (ResTime < Max_ResTime AND improve = true) do
    improve = false
    L1: forall T_ims ∈ R_jMS do
        L2: forall (R_j ∈ R AND R_j ≠ R_ms) do
            if (Move(T_ims to R_j) improves MS) then
                Move(T_ims, R_j); MS = Get_New_MS; improve = true;
                break L1;
            else
                L3: forall (T_i ∈ R_j AND R_j! = R_jMS) do
                    if Permut(T_i, T_ims) improves MS then
                        Permute(T_i, T_ims);
                        MS = Get_New_MS; improve = true;
                        break L1;
                    else
                    end
                end
            end
        end
    end
end
Return Task affectation;
```

Algorithm 5: *makespan* improvement

```
improve = true
L0: while (ResTime < Max_ResTime AND improve = true) do
    forall T_j1 ∈ R do
        L1: forall T_i1 ∈ R_j1 do
            if (!T_i1 ∉ Ts_Tested) then
                Ts_Tested.add(T_i1);
                L2: forall (R_j2 ∈ R AND R_j2 ≠ R_i1) do
                    if (Move(T_i1 to R_j2) improvesMS) then
                        Move(T_i1, R_j2); MS = Get_New_MS; improve = true;
                        break L1;
                    else
                        L3: forall (T_i2 ∈ R_j2) do
                            if Permut(T_i1, T_i2) improves MS then
                                Permute(T_i1, T_i2);
                                MS = Get_New_MS; improve = true;
                                break L1;
                            else
                            end
                        end
                    end
                end
            else
            end
        end
    end
end
Return Task affectation;
```

Algorithm 6: Task redistribution

4.6 Conclusion

The Heterogeneous Computing Scheduling Problem (HCSP) that we have addressed in this chapter is known to be an NP-Hard problem when it comes to optimize the *makespan*. The number of research works examining this problem keeps increasing, especially, with the increased need for computing power. The latter can be matched through powerful distributed computing architectures like Cloud, HPS, Fog, and Cloud computing.

In this chapter, we have presented two new task scheduling heuristics: *MMin* and *InterRC*. *MMin* is a fast deterministic heuristic, and *InterRC* is an evolutionary heuristic.

InterRC is an evolutionary heuristic that aims to evolve towards the better final *makespan* starting from an initial solution. It then switches alternatively between two phase (redistribution/improvement) until reaching one of two stop conditions already discussed above.

MMin combines the two well-known conventional heuristics *Min-Min* and *Max-Min*. Like the two latter, *MMin* is executed in n iterations, in each iteration one of the two heuristics is called (*Min-Min* or *Max-Min*) according to the global stat of the system.

All details about the experiments that we have conducted on our different proposals are presented in the next chapter (Chapter V).

CHAPTER V

Evaluation of the proposals

5.1 Introduction

In this chapter, we present the different experimentations that we have made to evaluate our three contributions described in the previous chapters. We start by presenting the InterRC evaluation phase in the first section, then, both the validation of different modules that compose *MFHS* and the evaluation of MMin algorithm will be presented in the same time in the second section. The latter is a little long compared to the first one, because it contains intensive experiments done in three different kinds of environments (simulator, emulator, and real Cloud based environment).

From InterRC point of view, the evaluation is done using our developed simulator. InterRC heuristic is compared with a set of non-evolutionary approaches, and with a set of evolutionary approaches. The main parameter studied in the evaluation phase is the total response time (*makespan*).

From the MFHS point of view, the part of our work that we present in this chapter allows to validate the different modules that compose our framework, while studying all QoS parameters discussed in Chapter III.

From *MMin* point of view, the evaluation phase that we discuss in this chapter shows the good result of *MMin* compared to two other heuristics, namely *Max-Min* and *Min-Min*. The evaluation of *MMin* is done using MFHS framework, which allows to both shows the good results mainly concern the total response time obtained using *MMin*, and validate the different modules that compose *MFHS* framework on the three environments cited below.

The three environments are used to highlight the reliability of *MFHS* (measured P_i=90% against 94% for the predicted P_i). Deployment and scheduling studies have also been achieved using MFHS as a simulator, *MFHS* in an experimental Cloud

based on *OpenStack*, and MFHS in an emulator test-bed based on *Emulab*. During experiments, four QoS parameters are taken into account (measured using *Resources Monitoring* module): energy consumption, cost, resource utilization, and makespan. These studies, as we mentioned before, are achieved using *MMin*, *Max-Min*, and *Min-Min* algorithms.

5.2 InterRC Evaluation

In order to evaluate *InterRC* approach, a simulator was developed using Java, R, and shell scripting. Then, the proposed *InterRC* algorithm was implemented and integrated to the developed simulator using the *Java* language. The evaluation phase was realized on a PC with Processor $i7$ and $8GO$ of RAM, on which, a set of experiments was done, then the results of the *InterRC* algorithm were compared with a set of algorithms, already presented in Chapter II.

5.2.1 Benchmark Datasets

In order to evaluate *InterRC* heuristic, we have used 12 classic problem instances proposed by Braun et al in [12], each instance having 512 tasks and 16 machines. An instance look like a 2D matrix composed 512 rows (or tasks) to be scheduled to 16 columns (or resource). Note that each element of this matrix denotes the expected execution time of a task on a resource.

The instances of these datasets are following u_x_yyzz structure, where u represents the uniform distribution that is used to generate these instances, x is the type of consistency, yy and zz are the task and resource heterogeneity respectively. The type of consistency is of the following types: consistent (c), inconsistent (i), and semi-consistent (s). The type of heterogeneity is either high (hi) or low (lo). Thus, the structure makes the following twelve instances: u_c_hihi, u_c_hilo, u_c_lohi, u_c_lolo, u_i_hihi, u_i_hilo, u_i_lohi, u_i_lolo, u_s_hihi, u_s_hilo, u_s_lohi, and u_s_lolo.

5.2.2 *makespan* Comparison

The proposed *InterRC* heuristic was compared with two kinds of heuristics: the first one is the set of fast deterministic heuristics, which are characterized by a low execution time and give the same result even if we repeat the execution of the algorithm many times. Meanwhile, the second kind is the evolutionary approaches

characterized by good results. The chosen fast deterministic heuristics are *RC* [122], *Sufferage* [77], *Min-Max* [53], and *Min-Min* [53]. Meanwhile, the used evolutionary heuristics are *cMA* [124], *GA* [42], *PA-CGA* [89] and *CHC* [29].

The execution of *InterRC* algorithm has been redone several times. The best obtained results are presented in Table 5.2 and Table 5.1. Table 5.2 shows the comparison of the *makespan* obtained by *InterRC* algorithm with the evolutionary algorithms, while Table 5.1 compares the obtained *makespan* with the best known and fast deterministic heuristics.

The last column (LP Bound) of two tables (Table 5.1 and Table 5.2) corresponds to the lower bound for the *makespan* value, which can be computed by solving the linear relaxation for the preemptive case using a linear programming solver [83]. The gray fields in both Table 5.1 and Table 5.2 indicate that the corresponding value is less than the value obtained by our algorithm, meaning that the *makespan* is not improved using our proposed algorithm.

Dataset	Min-Min	Min-Max	RC	Sufferage	LSufferage	InterRC	LP Bound
A.u_c_hihi.0	8460675.0	8205561.3	9576839.0	10249172.9	8092234.8	7434522,4	7346524.2
A.u_c_hilo.0	161805.4	161686.8	163200.2	168982.6	160100.3	154111,9	152700.4
A.u_c_lohi.0	275837.4	279907.7	309192.7	337121.5	255070.3	241582,7	238138.1
A.u_c_lolo.0	5441.4	5485.4	5542.6	5658.5	5487.4	5174,3	5132.8
A.u_i_hihi.0	3513919.3	3066454.8	3447651.4	3306818.9	3436518.1	2985279,8	2909326.6
A.u_i_hilo.0	80755.7	75711.6	76471.5	77589.1	77998.5	74194,2	73057.9
A.u_i_lohi.0	120517.7	108533.3	126002.4	114578.9	112400.9	104378,5	101063.4
A.u_i_lolo.0	2785.6	2613.5	2677.0	2639.3	2735.7	2569,4	2529.0
A.u_s_hihi.0	5160342.8	4627988.8	5068011.5	5121953.6	4394021.6	4216710,5	4063563.7
A.u_s_hilo.0	104375.2	100128.4	101739.6	102499.9	100813.8	97782,8	95419.0
A.u_s_lohi.0	140284.5	133039.3	143491.2	150297.1	134568.5	123327,7	120452.3
A.u_s_lolo.0	3806.8	3555.2	3679.6	3846.5	3695.8	3486,6	3414.8

Table 5.1: Makespan comparaison with deterministic heuristics

Instance	cMA	GA	PA-CGA	CHC	MA + TS	InterRC	LP Bound
A.u_c_hihi0	7700930	7659879	7437591	7599288	7530020	7434522,4	7346524.2
A.u_c_hilo0	155335	155092	154393	154947	153917	154111,9	152700.4
A.u_c_lohi0	251360	250512	242062	251194	245289	241582,7	238138.1
A.u_c_lolo0	5218	5239	5248	5226	5174	5174,3	5132.8
A.u_i_hihi0	3186665	3019844	3011581	3015049	3058475	2985279,8	2909326.6
A.u_i_hilo0	75857	74143	74477	74241	75109	74194,2	73057.9
A.u_i_lohi0	110621	104688	104490	104546	105809	104378,5	101063.4
A.u_i_lolo0	2624	2577	2603	2577	2597	2569,4	2529.0
A.u_s_hihi0	4424541	4332248	4229018	4299146	4321015	4216710,5	4063563.7
A.u_s_hilo0	98284	97630	97425	97888	97177	97782,8	95419.0
A.u_s_lohi0	130015	126438	125579	126238	127633	123327,7	120452.3
A.u_s_lolo0	3522	3510	3526	3492	3484	3486,6	3414.8

Table 5.2: Makespan comparaison with evolutionary heuristics

Three parameters can change the resulting *makespan*. The algorithm's execution time, the values of R_{j_init} and T_{i_init}, and the value $NbrIterations$. In our implementation, the fixed time for *InterRC* execution is $20s$ for all tests, R_{j_init} and T_{i_init} values are randomly selected in each test and the $NbrIterations$ is fixed to 4 each one (redistribution/improvement) executed for $5s$ to get $4 * 5 = 20s$, which is the total execution time dedicated to *InterRC*.

5.2.3 *makespan* Evolution Speed

In order to study the speed of the *makespan* evolution as a function of time when running the *InterRC* heuristic, the following process is followed:

The value of the best *makespan* ($best\_ms$) given by the *Min-Min*, *Max-Min*, *RC* and *Sufferage* algorithms is calculated before running the *InterRC* algorithm. Then, at each detection of improvement of the *makespan* value during *InterRC* execution, a ratio *ratio* between this *makespan* (ms_evol) and $best\_ms$ is calculated using Formula 5.1.

$$ratio = \begin{cases} \text{(best\_ms/ms\_evol)-1} & if\ \text{best\_ms} < \text{ms\_evol} \\ \text{1-(ms\_evol/best\_ms)} & else \end{cases} \tag{5.1}$$

A negative value of *ratio* means that $best\_ms$ is not yet reached, whereas a positive value of *ratio* means that ms_evol is better than $best\_ms$. The convergence of *raio* value to 0 means that ms_evol value converges to $best\_ms$ value. On the contrary, when the value of *ratio* keeps away from 0 to one of the other peak values (1 or -1), that means that the value of ms_evol, Keep away from the value of $best\_ms$. This move away leads to a better result if the peak value is 1, but in the other case, the move away leads to a worse result.

Figure 5.1 presents the evolution speed of *ratio* value obtained for each instance. It shows that the *makespan* improvement through *InterRC* is done quickly at first time, and the ms_evol value converges quickly towards the $BestMS$ value, but after some time, the improvement speed starts to become relatively heavy; ultimately, the *makespan* improvement can stop, which makes the continuation of the algorithm execution unnecessary.

A zoom of Figure 5.1 is presented in Figure 5.2. The latter gives more detail about the evolution speed of ms_evol before reaching the value of $best\_ms$. It is shows that the nature of this evolution can vary from one instance to another, which allows

to say that the evolution speed depends on the nature of tasks, as well as resources.

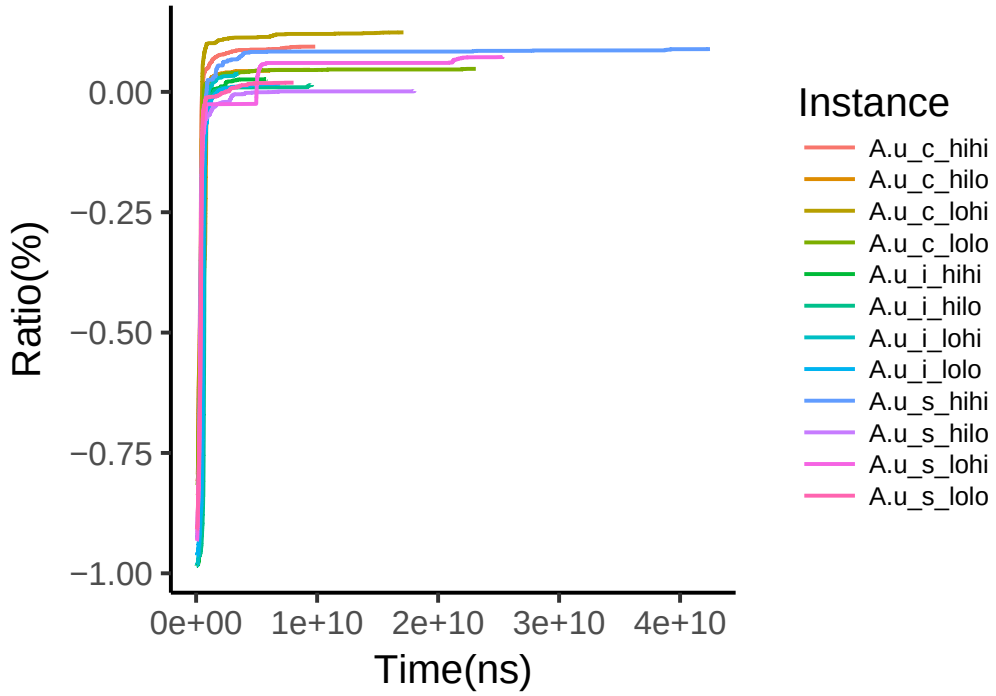


Figure 5.1: Evolution of *ratio* value as function of time

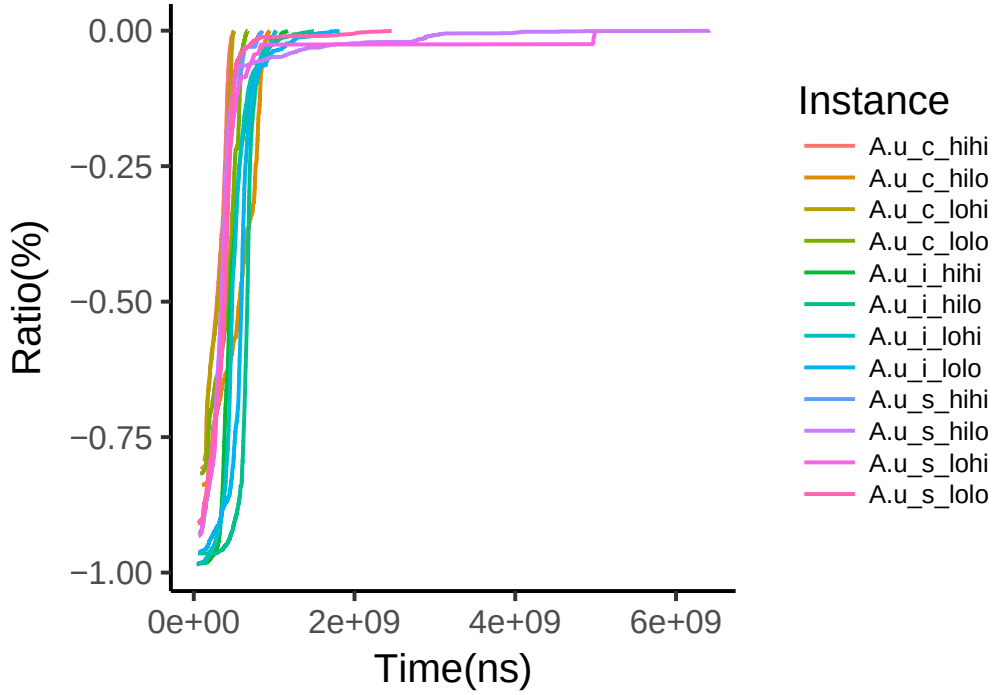


Figure 5.2: Zoom on Figure 5.1

The *gap* value is another parameter studied in our evaluation, it represents the relative gap value of any algorithm with respect to the corresponding lower bound, *gap* value is calculated using Formula 5.2. When MS_LP_Bound is the *makespan* obtained by lp bound and MS_Algo presents the *makespan* of the algorithm on which we look to calculate the *gap* value.

Figure 5.3 shows the average of *gap* value of the evaluated algorithms. The gap value of our proposed *InterRC* is the best one comparing with all other algorithms with a value equal to 2.013. Figure 5.3 shows also that the gap value of evolutionary approaches are the best comparing with all other fast deterministic heuristics.

$$gap = \frac{MS\_LP\_Bound - MS\_Algo}{MS\_LP\_Bound} \tag{5.2}$$

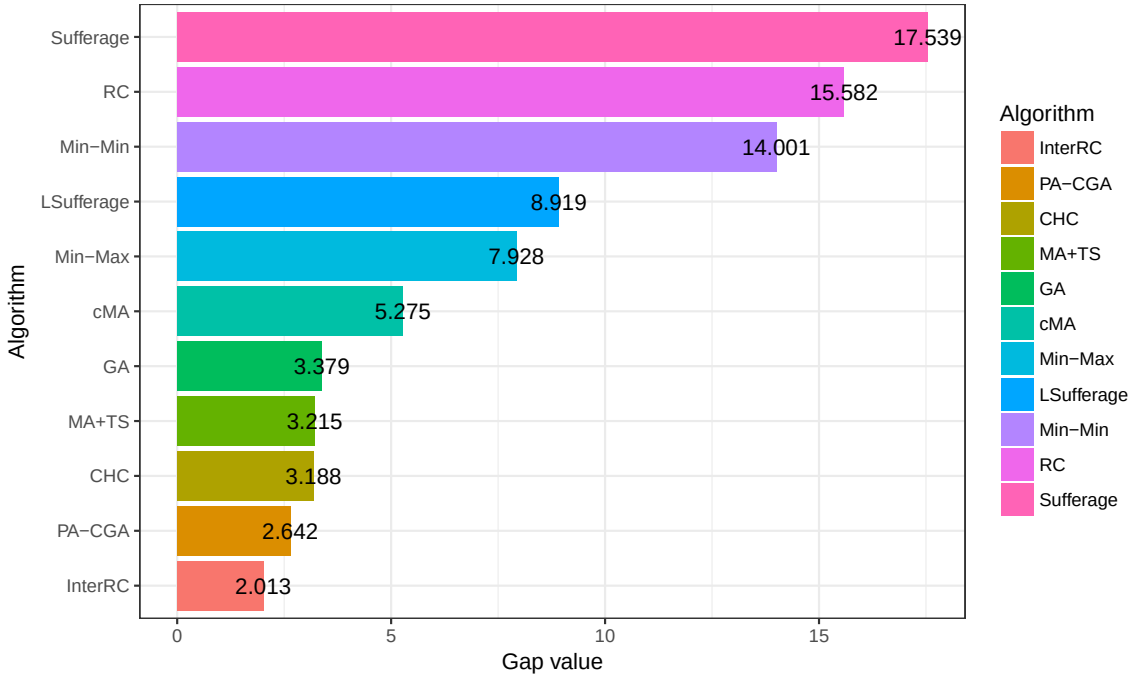


Figure 5.3: Gap value to the lower bound

5.3 MMin Evaluation and MFHS validation

This section aims to achieve two objectives, the first one is the *MMin* evaluation, and the second one is the *MFHS* validation. For these reasons, extensive experiences on *MMin* and *MFHS* are done on three different kinds of environments, which will be presented in detail in the remainder of this section.

5.3.1 Used Environments

The set of modules that compose our proposed *MFHS* were implemented using *Java*, *R*, and *Shell* programming languages. *MFHS* works in any Linux distribution based on Debian[1], like Ubuntu[2] on which we gave realized all our experimental studies.

The experimentation exposed in this section have been conducted using three different kind of environments, where *MFHS* is used as an evaluation tool. In the first one, *MFHS* was used as a theoretical tool on which an evaluation of the *MMin* algorithm is done. In the second one, a real *MFHS* framework experimentation and an evaluation of the *MMin* algorithm using real data are done on a test environment based on *Emulab* test-beds. Then, in the last one, a real *MFHS* framework experimentation, as well as the evaluation of the *MMin* algorithm are made in a production Cloud environment based on *OpenStack*.

In the following, we expose in detail the whole experimentation done in these three different contexts.

5.3.2 Input Data

In order to achieve our different experimentations, we need to introduce different data, concerning the task and resource characteristics.

For the resource characteristics, data are generated randomly for the simulated experimentation, whereas, all data are collected from a real environment for the experimentation done on an emulated based environment and for experimentation done on real Cloud based environment. The use of real data in the two latter kinds of experimentation gives more significance to our experimentations. The details of these data will be given during the presentation of each experimentation.

For the task characteristics, the data are generated randomly with a high heterogeneity between tasks. Similarly to resources, the different task characteristics will be presented in detail when presenting the corresponding experimentation.

[1]https://www.debian.org
[2]https://www.ubuntu.com

5.3.3 An illustrative example

In this subsection, a simple illustrative case is presented, where the heterogeneity of the set of candidate resources to a given task execution is clearly shown. Indeed, each task has a different execution time depending on the resource on which it is executed.

The following table shows the execution time of each task T_i on each resource R_i.

Task/Resource	R0	R1	R2
T0	5	8	10
T1	5	7	8
T2	9	3	6
T3	17	3	8
T4	14	12	28
T5	16	4	27

Table 5.3: Task Execution Time (s) on each resource

In the example shown in Table 5.3, the total *makespan* obtained with our proposed heuristic *MMin* is 17s, while the ones obtained with *Max-Min* and *Min-Min* are 21s and 19s, respectively.

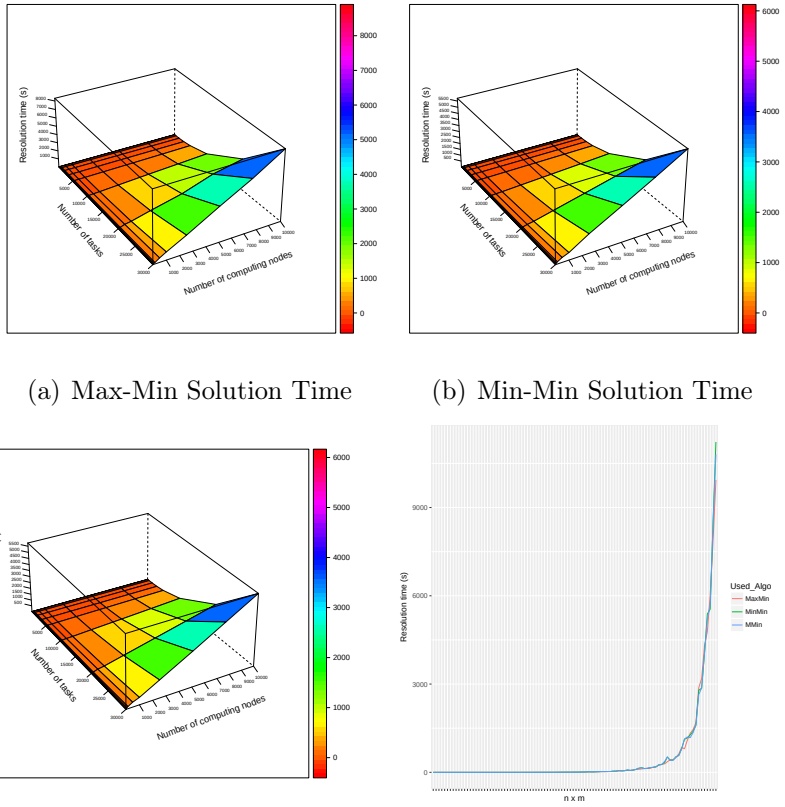
In order to clearly present the different possibilities of task allocation according to the resources taken into account, this example is very limited in terms of problem size (5 tasks and 3 resources).

5.3.4 Evaluation with simulated experimentation

Before moving to a real experimentation in a Cloud environment, a theoretical evaluation of the proposed heuristic has been done. Indeed, this type of evaluation is well-motivated due to the fact that the amount of resources as well as the number of tasks are not subject to cause any issue. In addition, it allows to make the evaluation much faster than proceeding the execution on a real environment.

As this section is dedicated to theoretical evaluation, it is easy to smoothly increase the allocation size problem leading to more complex cases to solve. Thus, the remainder of this subsection presents a kind of scalability evaluation up to a high amount of tasks and resources.

In order to evaluate and compare the proposed *MMin* with the original *Max-Min* and *Min-Min* algorithms in terms of *Solution Time*, the set of these algorithms was rewritten in Java Language. All measurements have been done on the same



(a) Max-Min Solution Time (b) Min-Min Solution Time

(c) MMin Solution Time (d) Solution Time comparison

Figure 5.4: 3D solution time illustration and comparison

server which is a *DELL R630* equipped with a *Intel(R) Xeon(R) CPU E5-2623 v3 @ 3.00GHz* running at its faster frequency (Turbo mod disable). Subsequently, *MFHS* is used to achieve the following study and allows to measure the *Resolution Time* in safe conditions for each algorithm.

This study has been done by smoothly varying the number of tasks and the number of resources. The values used for both tasks and resources are the following:

$$T = [\quad 30000 \quad 20000 \quad 10000 \quad 5000 \quad 3000 \quad 1500 \quad 500 \quad 100 \quad 50 \quad 10 \quad]$$

$$R = [\quad 10000 \quad 7500 \quad 5000 \quad 2500 \quad 1000 \quad 500 \quad 150 \quad 75 \quad 25 \quad 5 \quad]$$

This leads to get 100 cases, , each of which represents a point in the following 3D figures.

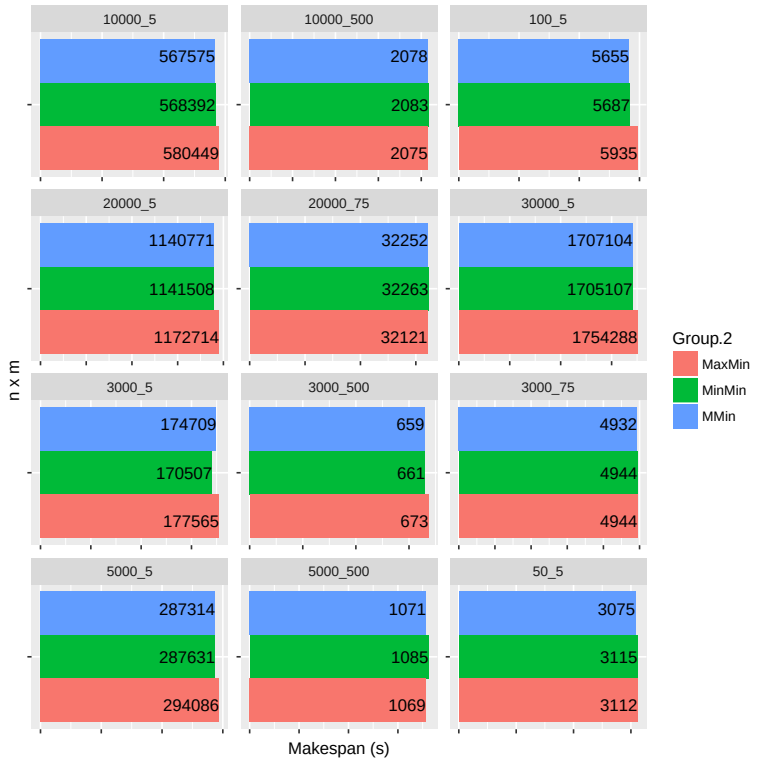


Figure 5.5: Makespan analysis with various problem sizes.

Figure 5.4 is composed of a set of sub-figures. In our case, this figure is composed of four sub-figures, where, Subfigure V.4(a), V.4(b) and V.4(c) give a three dimensional representation for each evaluated algorithm. The X and Y axes represent, respectively, the number of tasks $NbrT$ and the number of computing nodes $NbrR$, while the Z axis represents the time spent to find the scheduling solution. In Subfigure V.4(d), a comparison of these algorithms in terms of solution time is made. These graphs show that the combination of *Max-Min* and *Min-Min* to get the *MMin* algorithm has no influence on the resolution time of such a scheduling problem.

As several cases have been solved to generate the Figure 5.4, it is also possible to have a first result overview in terms of makespan. These results are highlight through the Figure 5.5, where only 12 cases have been selected (over the 100 cases done) and are illustrated 12 sub-figures. Each sub-figure correspond to a unique case ($NbrT$ x $NbrR$) which is indicated in the top, whereas the makespan result (in second) is indicated in the three bars corresponding to each of the three respective algorithms (*MinMin*, *MaxMin* and *MMin*). The experiments for all these cases have been conducted using a quite high difference between the task execution times. More precisely, short and long tasks have been defined in order to ensure heterogeneity. The

execution time for short tasks varies from 1s to 100s, whereas it varies from 1000s to 10000s for the long ones. The results presented in Figure 5.5 show that *MMin* almost gives the best makespan optimization. Indeed, for the whole 100 cases studied in this section, *MMin* was 37 times the best, *MinMin* 10 times and 13 times for *MaxMin*. In the other studied cases, the makespan is the same, especially when the number of resources is very large compared to the number of tasks.

5.3.5 Evaluation with Emulab

Emulab [49] is considered as one of traditional computing cluster environment that can be used as an experimental environment. Emulab allows to specify a network topology and link characteristics. Subsequently, it allows to get an heterogeneous environment suitable to make various experimentations and evaluations. In our thesis, we have used Emulab as an emulator of a Grid environment.

5.3.5.1 Resources Reservation

In the case of the *MFHS* experimentation, to evaluate *MMin*, *Max-Min* and *Min-Min* algorithms in an Emulab based environment, *MFHS* is deployed over an Emulab based environment using a set of 6 reserved physical machines connected through a private network with a star topology as shown in Figure 5.6.

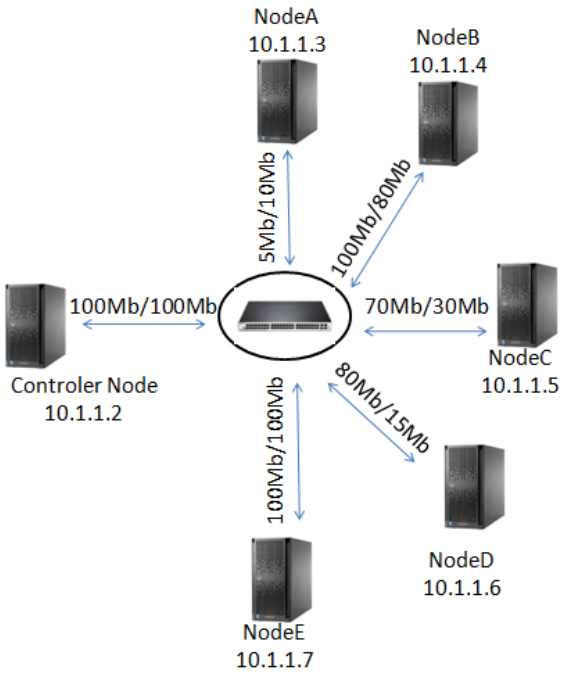


Figure 5.6: Topology

The different rates of the network cards of the set of used resources are fixed as indicated in Figure 5.6. Although, we let the *Resources Discovery* module to discover these rates values in order to show the right functioning of the *Resources Discovery* module and to get the Pi value defined by Formula 5.3.

As shown in Figure 5.6, one of the servers is dedicated to be a controller node, while the remaining five servers are used as computing nodes. *MFHS* is deployed on Controller node and the communication between the controller node and the computing nodes is made with SSH protocol using a Key Authentication Mechanism.

5.3.5.2 Resources Discovery

Using the *Resources Discovery* module, the collected information are described in Tables 5.4 and 5.5. Table 5.4 gives the download and upload throughputs between the controller node and the different computing nodes, while Table 5.5 gives the different write and read rate values.

Host	Upload (Mb/S)	Download (Mb/S)
10.1.1.3	1.10	0.56
10.1.1.4	7.17	8.53
10.1.1.5	3.11	6.53
10.1.1.6	1.66	4.88
10.1.1.7	8.40	7.71

Table 5.4: Download and Upload characteristics

Host	Buffer Disk Read (Mb/S)	Cached Disk Read (Mb/S)	Timing Write (Mb/S)
10.1.1.3	150.39	1940	98.0
10.1.1.4	74.97	1652	64.8
10.1.1.5	75.23	1664	66.0
10.1.1.6	60.10	2476	56.1
10.1.1.7	75.46	1628	66.5

Table 5.5: In/Out Disk speed characteristics

Knowing these values are real measured values (and not fixed ones), they could vary regarding physical or hardware equipment changes. Hence, the *Resources Discovery* module is in charge of sensoring these values before each scheduling operation.

5.3.5.3 Requests Collector

Algorithm 7 is used to test the *MFHS* approach on a real environment as well to evaluate and compare the *MMin* scheduling approach with the original *Min-Min* and *Max-Min* with real data.

Input : $UFS_i, DFS_i, RFS_i, WFS_i$
Output: ET_i, C_i, E_i
Call : $START$
Call : Upload(UFS_i) ; // Upload file of size UFS_i
Call : Download(DFS_i) ; // Download file of size UFS_i
Call : Read(RFS_i) ; // Read file of size UFS_i
Call : Write(WFS_i) ; // Write file of size UFS_i
Call : UseCPU($TimeCPU_i$,$EstimatedCPU_i$); // Use $EstimatedCPU_i$% of CPU during $TimeCPU_i$s
Call : Compute ET_i
Call : Compute C_i
Call : Compute E_i
Call : END
Return (ET_i, C_i, E_i)

Algorithm 7: Virtual machine execution scenario.

Assuming that each task T_i defined in Algorithm 7 is described as follows: $T_i = UFS_i, DFS_i, RFS_i, WFS_i$, where:

- UFS_i represents the size of the data to be uploaded
- DFS_i represents the size of data to be downloaded
- RFS_i represents the size of data to be read from disk
- FS_i represents the size of data to be written to disk

The requirements of each task are randomly generated. The generated values are listed in Table 5.6.

Task id	Upload FS (KB)	Download FS (KB)	Read FS (KB)	Write FS(KB)	vCPUs number	CPU Time(S)
T0	15000	25000	18000	25000	2	340
T1	20000	15000	15000	40000	2	180
T2	15000	15000	25000	12000	2	100
T3	55000	10000	11000	50000	2	210
T4	32000	9000	15000	19000	2	70
T5	32000	18000	5000	39000	2	90
T6	8000	25000	30000	40000	2	300
T7	50000	17000	12000	77000	2	80
T8	60000	19000	11000	12000	2	200
T9	80000	5000	20000	7000	2	60
T10	25000	10000	80000	10000	2	210
T11	52000	25000	32000	46000	2	140
T12	10000	10000	30000	20000	2	280
T13	35000	14000	50000	15000	2	80
T14	50000	50000	21000	24000	2	160
T15	92000	10000	8000	41000	2	260
T16	55000	15000	15000	40000	2	100
T17	15000	10000	35000	50000	2	190
T18	57000	9000	15000	19000	2	150
T19	20000	35000	5000	55000	2	120
T20	60000	5000	15000	39000	2	100
T21	25000	17000	22000	35000	2	80
T22	20000	9000	21000	22000	2	120
T23	12000	5000	60000	7000	2	130
T24	15000	7000	120000	10000	2	150
T25	12000	5000	30000	28000	2	200
T26	10000	10000	30000	65000	2	200
T27	40000	5000	75000	77000	2	180
T28	15000	250000	92000	56000	2	160
T29	8000	18000	20000	40000	2	90

Table 5.6: Dynamic: Requests Description

5.3.5.4 Scheduling

After the execution of the *Resources Discovery* and *Requests Collector* modules, the scheduling process starts. Running the chosen algorithms and waiting for their

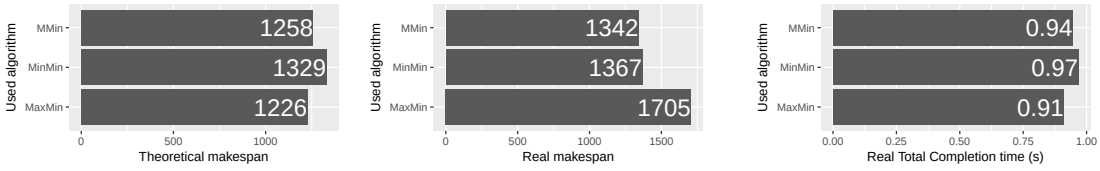
execution end, it allocates the tasks to different resources. Assignment of tasks is described in Table 5.7. To be able to compare the algorithms, the scheduler is launched with three different algorithms, namely *MMin*, the original *Max-Min* and the original *Min-Min*.

Algorithm	Max-Min	Min-Min	MMin
Host	Tasks	Tasks	Tasks
R0	T12,T17,T20,T23,T3	T1,T15,T16,T26,T29	T12,T16,T17,T20,T3
R1	T0,T11,T19,T21,T27,T5	T10,T19,T2,T21,T27,T28	T0,T11,T19,T2,T27,T4
R2	T13,T14,T18,T22,T26,T28,T4	T11,T14,T22,T3,T4,T6,T7	T14,T18,T23,T26,T28,T29,T5
R3	T10,T2,T24,T25,T6	T0,T13,T17,T20,T24,T8	T10,T21,T24,T25,T6
R4	T1,T15,T16,T29,T7,T8,T9	T12,T18,T23,T25,T5,T9	T1,T13,T15,T22,T7,T8

Table 5.7: Dynamic: Tasks allocation

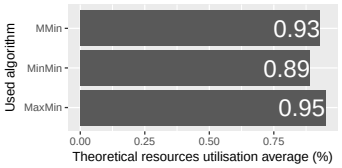
5.3.5.5 Behavior Study

During the task execution process, it is possible to monitor the progress of this process with the *Resources Monitoring* module. This module is able to detect the failures related to the network, the disks or the processing. Then the state of this process is communicated to the *Behavior Study* module. In our evaluation, after the end of execution of all tasks with success, we get the results depicted in a set of figures which also are automatically generated from the *Behavior Study* module. Figures 5.7, 5.8, 5.9 and 5.10 depict all the results per algorithm. Each of these figures is composed of three sub-figures. The first one exposes the results obtained through the theoretical analysis, the second one gives the real results obtained after the end of execution, and the last one gives a ratio between these two. In another set of figures, (figures 5.11, 5.12, 5.13 and 5.14), the results are illustrated in more detail by representing the completion time, resource utilization, energy and cost metrics' values for each resource used.

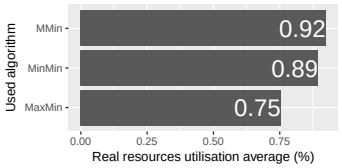


(a) *Emulab:* Theoretical makespan (b) *Emulab:* Real makespan (c) *Emulab:* Real vs Theoretical makespan

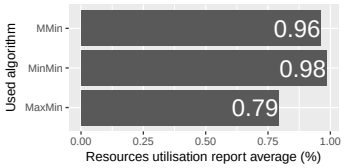
Figure 5.7: Theoretical vs Real makespan



(a) *Emulab:* Theoretical Resources utilization average

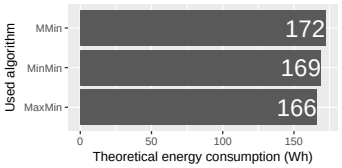


(b) *Emulab:* Real Resources utilization average

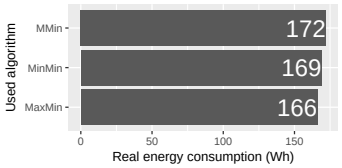


(c) *Emulab:* Real vs Theoretical Resources utilization average

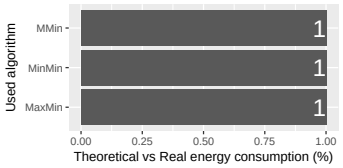
Figure 5.8: Theoretical vs Real Resources utilization average per algorithm



(a) *Emulab:* Theoretical energy consumption per algorithm

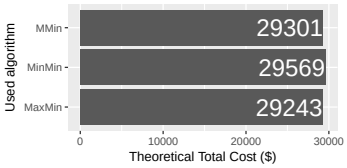


(b) *Emulab:* Real energy consumption per algorithm

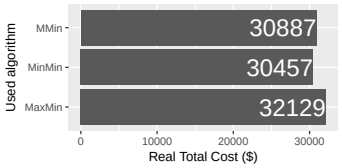


(c) *Emulab:* Real vs Theoretical energy consumption per algorithm

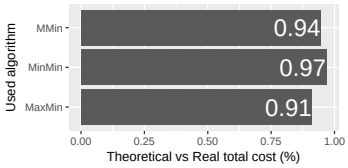
Figure 5.9: Theoretical vs Real energy consumption per algorithm



(a) *Emulab:* Theoretical Cost consumption per algorithm

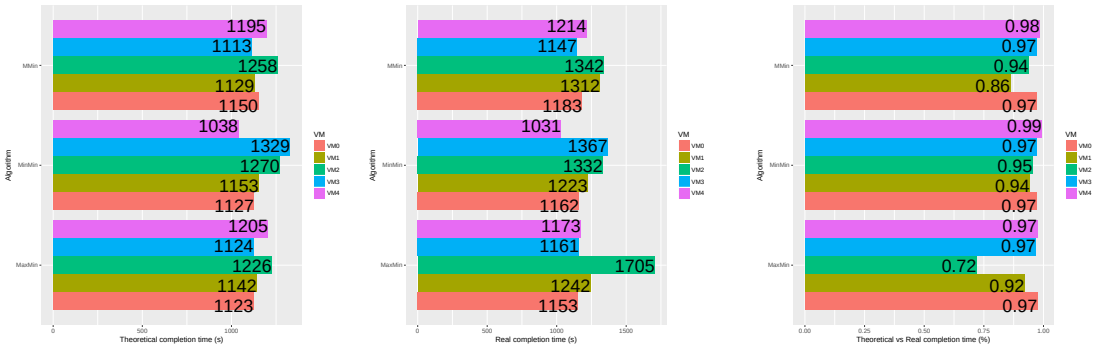


(b) *Emulab:* Real Cost per algorithm



(c) *Emulab:* Real vs Theoretical cost per algorithm

Figure 5.10: Theoretical vs Real Cost per algorithm

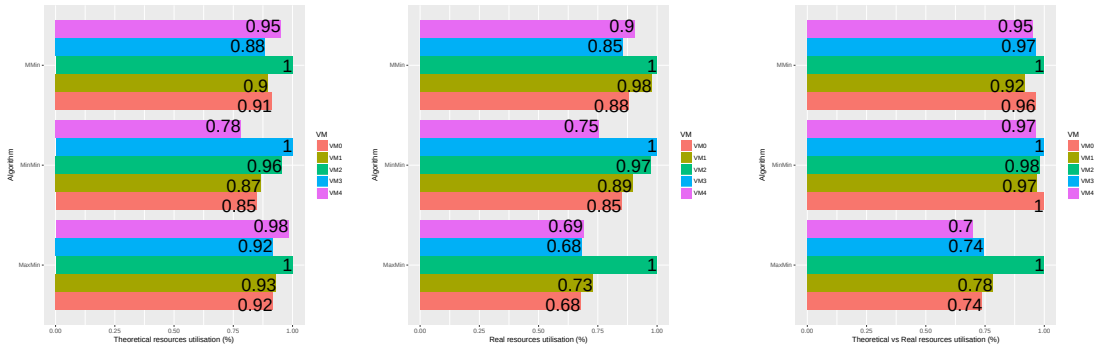


(a) *Emulab:* Theoretical Completion time per VM per Algorithm

(b) *Emulab:* Real Completion time per VM per Algorithm

(c) *Emulab:* Real vs theoretical Completion time report per VM per Algorithm

Figure 5.11: Theoretical vs Real Completion time per VM per algorithm

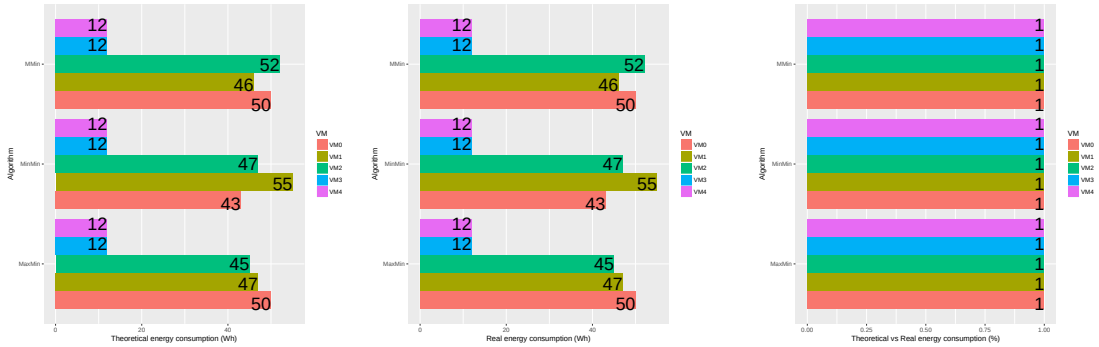


(a) *Emulab:* Theoretical Resources utilization per VM per Algorithm

(b) *Emulab:* Real Resources utilization per VM per Algorithm

(c) *Emulab:* Real vs Theoretical Resources utilization per VM per Algorithm

Figure 5.12: Theoretical vs Real Resources utilization per VM per algorithm

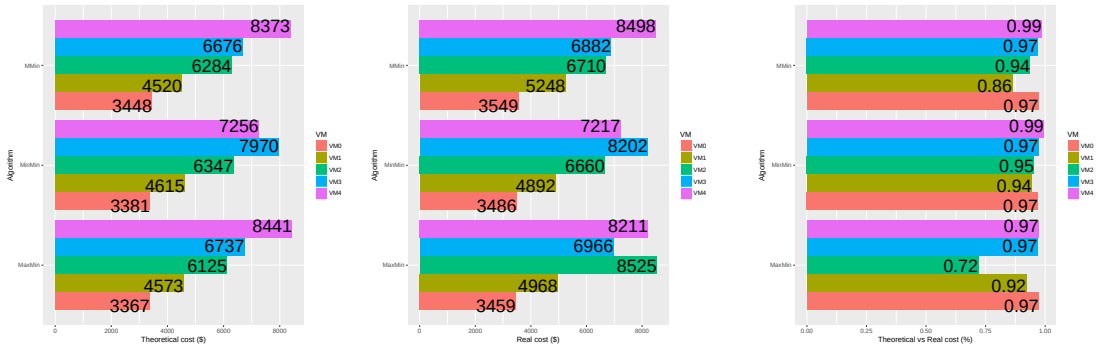


(a) *Emulab:* Theoretical Energy per VM per Algorithm

(b) *Emulab:* Real Energy per VM per Algorithm

(c) *Emulab:* Real vs Theoretical Energy per VM per Algorithm

Figure 5.13: Theoretical vs Real Energy per VM per algorithm



(a) *Emulab:* Theoretical Cost per VM per Algorithm (b) *Emulab:* Real Cost per VM per Algorithm (c) *Emulab:* Real vs Theoretical Cost per VM per Algorithm

Figure 5.14: Theoretical vs Real Cost per VM per algorithm

5.3.6 Experimentation on real experimental Cloud Platform

To validate the proposed *MFHS* framework and evaluate the proposed scheduling approach *MMin*, an experimental Cloud Computing platform based on *OpenStack* (Kilo version) has been used. First of all, we studied the reliability of the *extra_specs* module used to manage the input/output rates of the network card and the read/write disk rates. This phase is done in order to find an index of prediction denoted Pi that allows to estimate the proportion of convergence between the theoretical and real execution. Pi will be calculated during the *Resources Discovery* process.

5.3.6.1 Resources Reservation

To build an experimental environment based on *MFHS* framework, firstly, the *MFHS* controller node is deployed on existing cloud environment based on *Openstack*. The deployment is done on the same node that contains the main Cloud controller node. Secondly, a set of 6 VM is deployed on the Cloud environment to be used as the *MFHS* compute nodes.

As described in Table 5.8, each VM has its own characteristics in terms of disk read/write rates and data network transfer rate. As mentioned above, the resource heterogeneity is obtained by setting up the *extra_specs OpenStack* module.

Host	Upload (Kb/S)	Download (Kb/S)	Write Disk (Mb/S)	Read Disk (Mb/S)
VM0	256	512	5	7
VM1	256	512	2	2
VM2	512	1024	8	4
VM3	128	1024	6	8
VM4	256	256	5	7
VM5	2048	128	3	3

Table 5.8: Resources characteristics

5.3.6.2 Resources Discovery

In order to find the Pi, we have used two popular Linux programs: *scp* and *dd*. The *scp* program allows to make a secure transfers of files between the controller node and any compute node and *dd* allows to read or write data on disk. Consequently, using these two programs allowed us to check the efficiency and reliability of the *extra_specs* module regarding these network and disk rates.

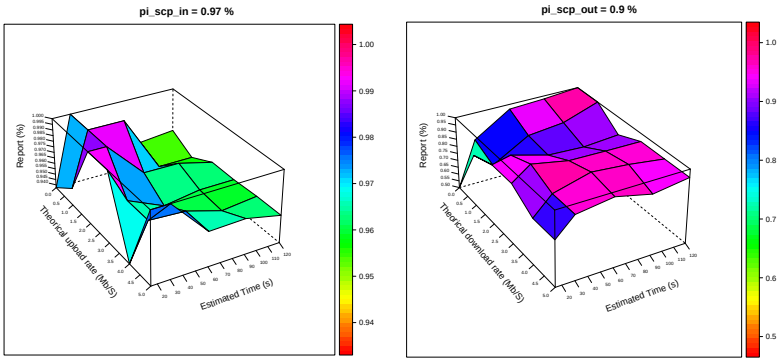
The tests have been conducted for the following cases:

- *Upload*: This test allows computing the efficiency degree (pi_scp_out) of the module that limits the throughput of the output card.
- *Download*: This test allows computing the efficiency degree (pi_scp_in) of the module that limits the throughput of the input card.
- *Disk read*: This test allows computing the efficiency degree (pi_disk_read) of the module that limits the reading rate from disk.
- *Disk write* This test allows computing the efficiency degree (pi_disk_write) of the module that limits the write rate on disk.

We remind the reader that these testing steps have been done locally on each instantiated VM, meaning the Upload term is associated to the amount of data sent from the network car of each VM, and the Download term is associated to the amount of data received by the network card of each VM.

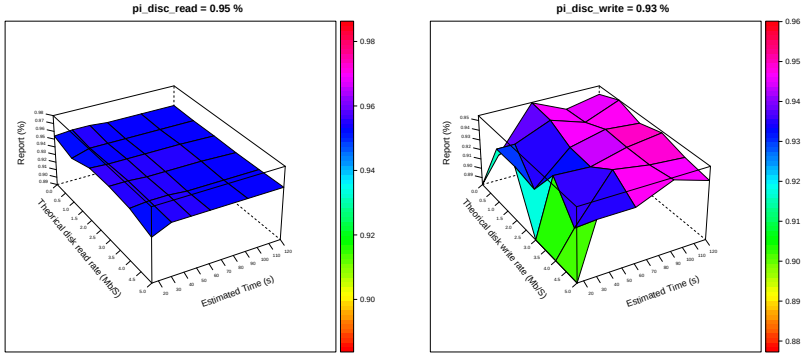
To get the P_i value, a set of intensive experiments was carried-on using the *MFHS* framework, results are illustrated by the two figures 5.15 and 5.16.

Figure 5.15 includes four sub-figures V.15(a) V.15(b) V.15(c)and V.15(d), each one corresponding to one of the four partial experiences which allows to have a partial Pi. Then, Formula 5.3 is used to get the global Pi, while each partial Pi denoted $PartialP_i$ is computed using Formula **??**. The set of partial P and the global P will



(a) *OpenStack:* Scp In measurements

(b) *OpenStack:* Scp Out measurements



(c) *OpenStack:* Disk read measurements

(d) *OpenStack:* Disk write measurements

Figure 5.15: 3D Data rate transfer measurements

be used by the other modules of *MFHS*. In Figure 5.15, the x axis represents the estimated test time of the corresponding theoretical rate represented by the y axis. Meanwhile, the z axis represents the ratio between theoretical and real time obtained after a real experience. Note that the ratio of the *sth* experience, denoted $Report_i$, is computed as follow: Assume that the theoretical throughput is TD and the real throughput computed by launching a real test on the reserved resource is RD. Then $Report_i$ is computed using the Equation 5.5, while the $PartialP_i$ of each of the four operations (scp in, scp out, disk read and disk write) is obtained by calculating the sum of the $Report_i$ obtained after each test divided on the number of the tests. Each of these $PartialP_i$ is mentioned on the associated sub-figure.

$$Pi = \frac{pi\_scp\_in + pi\_scp\_out + pi\_disc\_read + pi\_disc\_write}{4} \tag{5.3}$$

$$PartialP_i = \frac{\sum_{i=1}^{n} Report_i}{n}, \quad PartialP_i \in \{pi\_scp\_in, pi\_scp\_out, pi\_disc\_read, pi\_disc\_write\} \tag{5.4}$$

$$Report_i = \frac{Min(T_t, R_t)}{Max(T_t, R_t)} \tag{5.5}$$

From sub-figures V.15(a), V.15(b), V.15(c) and V.15(d), it is clear that the *extra_specs* module is reliable, since the majority of the performed tests show a report value greater than 0.94.

Figure 5.16 shows a real time evolution of data transfer rate processes. Each sub-figure allows following the evolution in real time of the four particular operations observed: Upload (Scp Out), Download (Scp In), Disk read and Disk write. The horizontal lines correspond to theoretical Upload and Download rates which as set-up for each resource (Table 5.8). In the Download rate figure, only four horizontal lines are shown as this rate is respectively the same for $R1$ / $R2$ (512Kb/S), and for $R4$ / $R4$ (1024Kb/S). The real Download rates monitored during an *scp in* from each virtual machine all start with a rate of about 3000Kb/s and then clearly decrease down to the chosen value. It is also interesting to note that each real measured rate is quite stable over the time, except for a few peaks down on only one resource.

The top left part of Figure 5.16 shows in real time the rate evolution for the data upload to the desired resource, it is remarkable that the upload rate in each experience start from 4Mb/s, that's it quickly converges to the theoretical corresponding rate, then, the rate stays stable during the rest of the data transfer. In this figure six curves and four horizontal lines are shown, while six different resources are used in this experience. This is explained by the fact that $R0$, $R1$ and $R4$ have the same theoretical rate as shown in Table 5.8.

The bottom left part of Figure 5.16 represents the real time monitoring of the data read from the storage disk. It has a behavior similar to the top left part of the figure which depicts the upload rate experience, except that the rate starts with a high value close to 4Gb/s, and the convergence to the theoretical corresponding value is slower. Then, the bottom right part represents the experiment regarding the real time monitoring of the write rate on the resource's storage disk. The figure shows that the rate starts with a value close to the theoretical fixed rate, then, the rate it slightly varies during the experiment.

We conclude that the behaviors of the experiments are different from each other,

but the $extra_spec$ module is reliable in all the experimentation. These figures remain important to detect anomalies that may arise during the execution of such a data transfer or I/O operation.

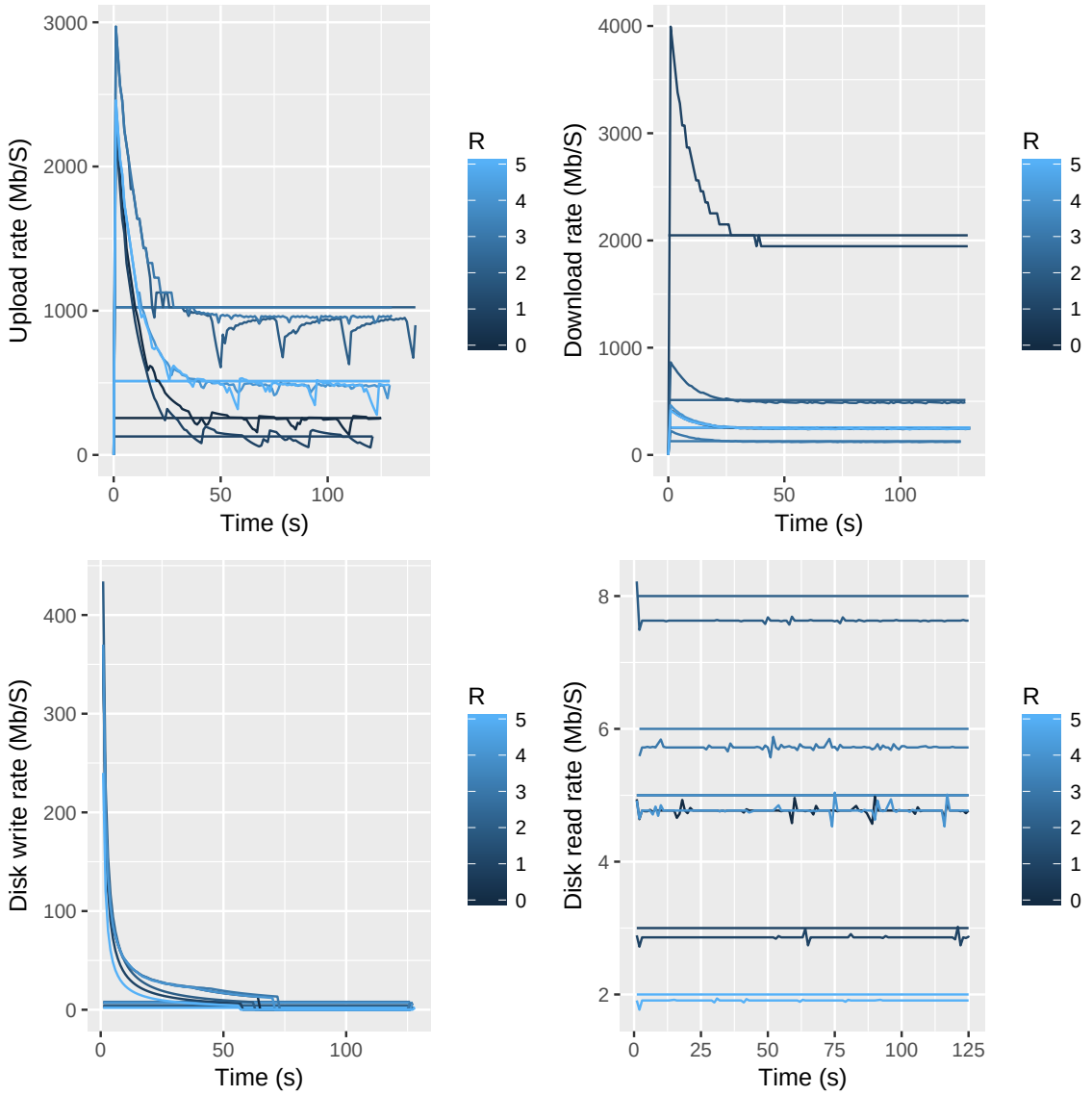


Figure 5.16: Real time data rate transfer monitoring

5.3.6.3 Requests Collector

The program 7 defined in Subsection 5.3.5.3, is still used for the experiments on *OpenStack* environment. The following table describes the characteristics of the set of tasks on which the experimentation and evaluation of the algorithm is carried out:

Task id	Upload FS (KB)	Download FS (KB)	Read FS (KB)	Write FS(KB)	vCPUs number	CPU Time(S)
T0	1000	2000	18000	11000	6	440
T1	2000	5000	15000	40000	6	180
T2	1000	15000	25000	12000	6	40
T3	6000	10000	11000	50000	6	110
T4	3000	9000	15000	19000	6	80
T5	12000	8000	5000	39000	6	90
T6	8000	5000	30000	40000	6	300
T7	5000	7000	12000	77000	6	80
T8	8000	9000	11000	3000	6	100
T9	12000	5000	20000	7000	6	150
T10	25000	10000	80000	10000	6	110
T11	12000	25000	32000	46000	6	140
T12	10000	10000	30000	20000	6	180
T13	4000	4000	50000	15000	6	110
T14	5000	5000	21000	24000	6	60
T15	1000	1000	8000	41000	6	260
T16	2000	15000	15000	40000	6	100
T17	6000	10000	35000	50000	6	190
T18	3000	9000	15000	19000	6	150
T19	12000	8000	5000	55000	6	120
T20	8000	5000	15000	39000	6	100
T21	5000	7000	22000	35000	6	80
T22	2000	9000	21000	22000	6	120
T23	12000	5000	60000	7000	6	130
T24	15000	7000	120000	10000	6	150
T25	12000	5000	30000	28000	6	200
T26	10000	10000	30000	65000	6	200
T27	4000	5000	75000	77000	6	180
T28	15000	15000	92000	56000	6	160
T29	8000	8000	20000	40000	6	90

Table 5.9: Openstack:Requests Description

5.3.6.4 Scheduling

In the first step, a filter is applied to select the set of resources that will participate to the execution of the set of tasks. After defining the detailed description of each task, any scheduling algorithm can be used. In our case, we call the *MMin*, *Max-Min* and *Min-Min*, each of these algorithms returns as output the assignment of tasks to different reserved resources.

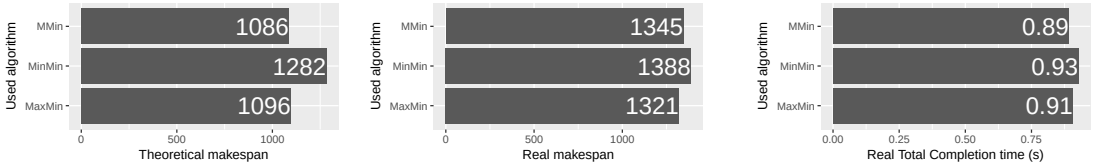
The following table gives us the task assignment returned by each algorithm:

Algorithm	Max-Min	Min-Min	MMin
Host	Tasks	Tasks	Tasks
R0	T12,T17,T20,T23,T3	T1,T15,T16,T26,T29	T12,T16,T17,T20,T3
R1	T0,T11,T19,T21,T27,T5	T10,T19,T2,T21,T27,T28	T0,T11,T19,T2,T27,T4
R2	T13,T14,T18,T22,T26,T28,T4	T11,T14,T22,T3,T4,T6,T7	T14,T18,T23,T26,T28,T29,T5
R3	T10,T2,T24,T25,T6	T0,T13,T17,T20,T24,T8	T10,T21,T24,T25,T6
R4	T1,T15,T16,T29,T7,T8,T9	T12,T18,T23,T25,T5,T9	T1,T13,T15,T22,T7,T8,T9

Table 5.10: Openstack:Tasks affectation

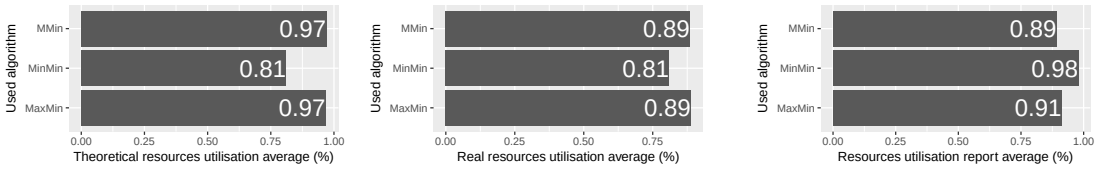
5.3.6.5 Behavior Study

Using the responsive module of the behavior study defined in *MFHS*, a set of 8 figures (Figure 5.17 to Figure 5.24) has been automatically generated at the end of all scheduling process. In the following, all those different figures are discussed with an explanation of the meaning of each one. In addition, the results obtained for each evaluated algorithm in this experimentation are compared.



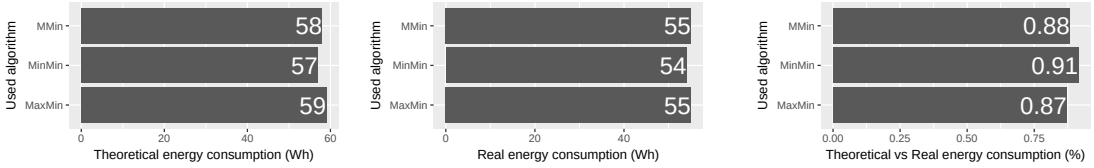
(a) *OpenStack:* Theoretical makespan (b) *OpenStack:* Real makespan (c) *OpenStack:* Real vs Theoretical makespan

Figure 5.17: Theoretical vs Real makespan



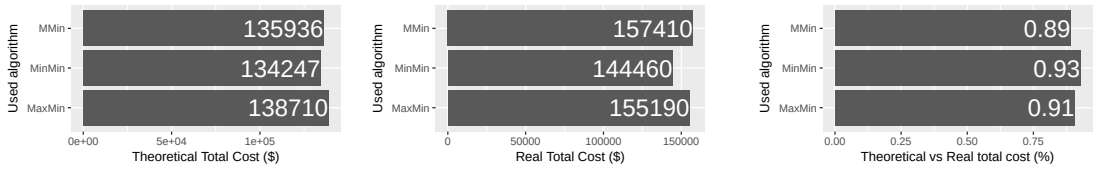
(a) *OpenStack:* Theoretical Resources utilization average (b) *OpenStack:* Real Resource utilization average (c) *OpenStack:* Real vs Theoretical Resource utilization average

Figure 5.18: Theoretical vs Real Resources utilization average per Algo



(a) *OpenStack:* Theoretical energy consumption per algorithm (b) *OpenStack:* Real energy consumption per algorithm (c) *OpenStack:* Real vs Theoretical energy consumption per algorithm

Figure 5.19: Theoretical vs Real energy consumption per algorithm

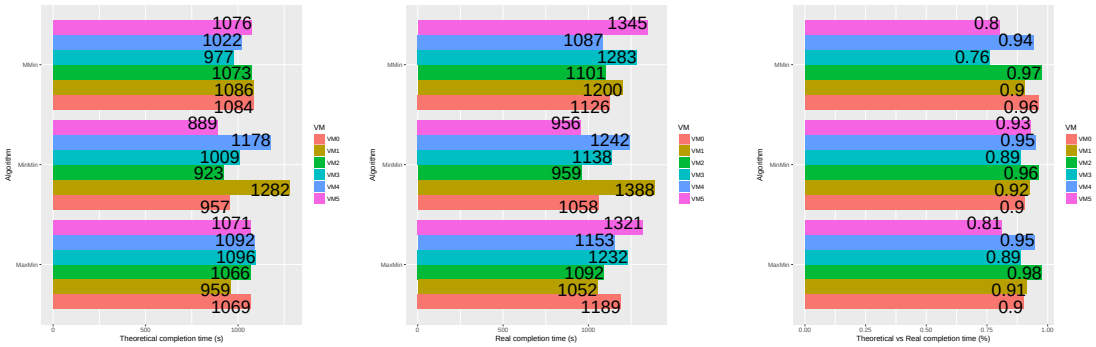


(a) *OpenStack:* Theoretical Cost per algorithm

(b) *OpenStack:* Real Cost per algorithm

(c) *OpenStack:* Real vs Theoretical cost per algorithm

Figure 5.20: Theoretical vs Real Cost per algorithm

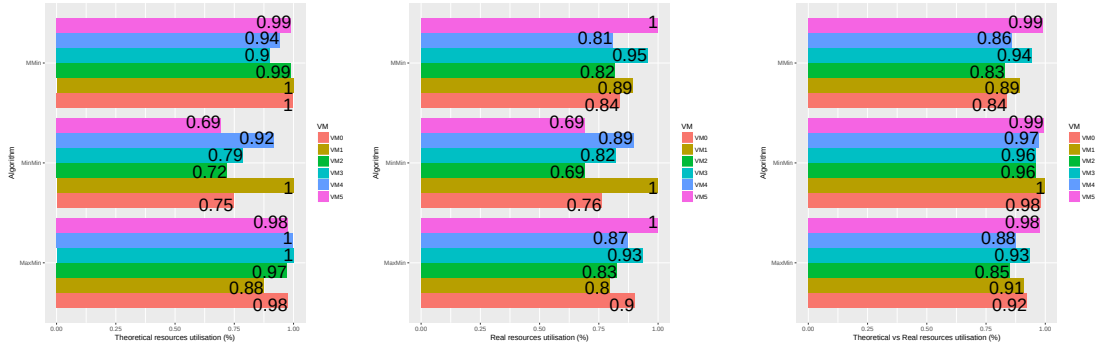


(a) *OpenStack:* Theoretical Completion time per VM per Algorithm

(b) *OpenStack:* Real Completion time per VM per Algorithm

(c) *OpenStack:* Real vs theoretical Completion time report per VM per Algorithm

Figure 5.21: Theoretical vs Real Completion time per VM per algorithm

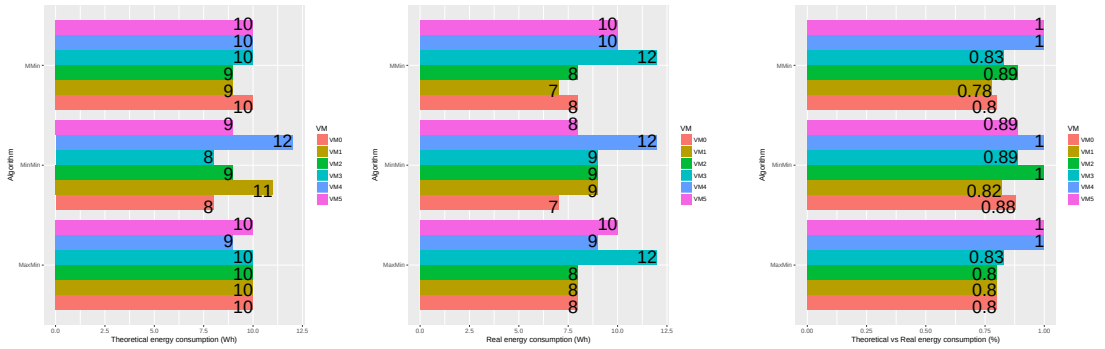


(a) *OpenStack:* Theoretical Resources utilization per VM per Algorithm

(b) *OpenStack:* Real Resources utilization per VM per Algorithm

(c) *OpenStack:* Real vs Theoretical Resources utilization per VM per Algorithm

Figure 5.22: Theoretical vs Real Resources utilization per VM per algorithm

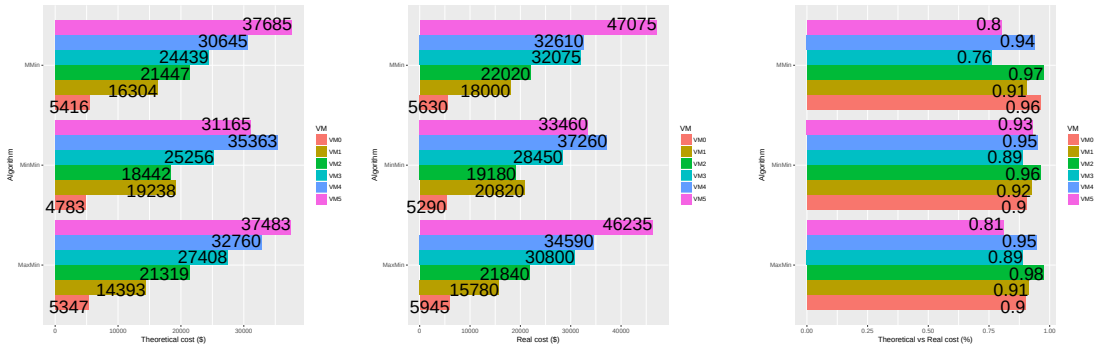


(a) *OpenStack:* Theoretical Energy per VM per Algorithm

(b) *OpenStack:* Real Energy per VM per Algorithm

(c) *OpenStack:* Real vs Theoretical Energy per VM per Algorithm

Figure 5.23: Theoretical vs Real Energy per VM per algorithm



(a) *OpenStack:* Theoretical Cost per VM per Algorithm

(b) *OpenStack:* Real Cost per VM per Algorithm

(c) *OpenStack:* Real vs Theoretical Cost per VM per Algorithm

Figure 5.24: Theoretical vs Real Cost per VM per algorithm

Figure 5.17, Figure 5.18, Figure 5.19 and Figure 5.20 give a global view of the behavior of each of the used scheduling algorithms, while Figure 5.21, Figure 5.22, Figure 5.23 and Figure 5.24 give a detailed view of the behavior of the corresponding used scheduling algorithm. On each of these figures, the left sub-figure depicts the theoretical values obtained before starting the real execution of the corresponding algorithm on real Cloud based on OpenStack, the middle one depicts the values obtained on the real Cloud platform which are obtained after the end of execution of the corresponding algorithm, finally the right sub-figure shows the computed ratio between these two.

Over the set of figures, we can say that *MMin* gives the best makespan, followed by the Max-Min, and then Min-Min as described in Figure V.17(a). Meanwhile, *Min-Min* ensures the best result with regard to consumed energy (Figure V.19(a)) and

cost (Figure V.20(a)). Finally, *Max-Min* and *MMin* are the best in same position concerning resources utilization average according to Figure V.18(a).

In the right sub-figure in all figures, a ratio between the theoretical values obtained on the real Cloud platform is computed. It shows the efficiency of the used tools in the execution, especially with regard to disk read/write speeds and network upload/download speeds. In all cases, we show that the measured value Pi is great than 88%, with an average of 91.25% for all experimentations.

5.4 Discussion

For InterRC heuristic, our experiments are conducted through simulation, where different comparisons of the proposed *InterRC* heuristic with others heuristics show that the proposed approach gives a better *makespan* in about 90 % of cases comparing with evolutionary approaches, and in 100 % of cases comparing with fast deterministic heuristics.

For MMin Evaluation and MFHS validation, the different experiments were conducted in three different environments. The first one consists to use a personal computer, which allows to demonstrate the possibility of use of *MFHS* to make theoretical analysis. The second one consists to use an emulator that emulates a distributed environment, demonstrating the possibility of using *MFHS* in an emulated environment. The last one consists to use a real experimental cloud environment, demonstrating the possibility of use of *MFHS* in a real distributed environment. Note that all experiments in the three different environments have been done to show the proper functioning of *MFHS* while taking advantage of the opportunity to compare our proposed *MMin* algorithm with the original *Max-Min* and *Min-Min* algorithms. This just to give an example, as well as it is possible to make the different experimentations with other scheduling algorithms and in other distributed environments. In our experiments, we have considered response time, load balancing, cost, and energy consumption as QoS parameters.

Looking at the QoS parameters measured in the *OpenStack* environment, it is well observed that resources utilization using *Max-Min* and *MMin* are relatively high compared to *Min-Min*, while this latter consumes less energy and cost compared to *MMin* and *Max-Min*. Consequently, it is recommended to use *Min-Min* for cost and energy optimization, *MMin* for makespan optimization, and to choose between *MMin* and *Max-Min* to optimize the resources utilization parameter.

From the experiments conducted on *Emulab*, it is clear that the makespan obtained

using *MMin* is always better than the other two algorithms, even for resource usage. We also notice that the results are similar to the one obtained in the *Openstack* environment. This means that the *Min-Min* algorithm does not exploit the resources very well compared to the other two algorithms. However, by taking a look at the energy consumption and cost, we can notice that the results are not similar in the *Emulab* and *Openstack* platforms. On the one hand, the energy consumed using *Max-Min* algorithm is less than the other two algorithms. Concerning the cost parameter, the *MMin* heuristic gives the best result which is the lowest compared to the cost obtained using *Min-Min* or *Max-Min*. This result difference in both *Emulab* and *Openstack* experimental environments can be explained by the fact that the resource utilization, cost, and energy parameters are not subjected to be optimized by the *MMin* heuristic. On the other hand, the makespan metric, which is intrinsically optimized by the proposed *MMin*, always gives the best result using the latter in both experimental environments.

For both *OpenStack* and *Emulab* environments, the experiments start by a theoretical analysis, then a real deployment is done in the second step, while computing the Pi, which allows to predicate the ratio between the results obtained with a theoretical analysis, and the ones that will be obtained after a real deployment. In both environments, the fact of having a convergence close to 90% between real results and theoretical ones, in most of the cases for all computed QoS parameters, shows the efficiency of all *MFHS* component modules. Also, the computed offset of 90% is very close to the predicted Pi which was about 94%. This result shows us that the experimentation has been set using relevant tools for managing network and disk rate and shows that the *OpenStack extra_space* module worked pretty well during the whole real virtual machine execution.

All *MFHS* modules have been explored, from resources discovery and requests collector to resources allocation, resources monitoring, and behavior study, while passing by exploring scheduling module.

Also, in both used environments, the transition from theoretical analysis to a real deployment is done automatically with any code source modification. Moreover, the transition from the environment to another is done exploring the modularity property of *MFHS* which facilitates the reuse of modules.

The different realized experiments allow us to say that *MFHS* functions are reached as intended. Hence, all modules are portable and easily reusable in various distributed environments.

One of the highlighted aims in presenting*MFHS* is to propose a modular framework which can be used in both theoretical, virtual and real environments. To demonstrate these facilities, we chose to expose studies in these 3 cases. Doing this, we hope to clearly show that good accuracy results can be reached, even on theoretical and *Emulab* environments which do not require long and hard installation and configuration. And so, it is not necessary to devote much time to setting up experiments.

Evaluating *MFHS* in different experimental setups are also important in order to face with:

- The reliability of models used
- Multiple levels of external perturbation
- Multiple levels of platform accuracy
- Multiple difficulty levels to propose reproducible experimentations
- Highlight *MFHS* can be used in a reliable way in different experimental setups, and able to switch from one to another.

About the framework performance, it is not affected while moving from theoretical studies to emulation and from emulation to real cloud testbed. Obviously, for the reasons described above, it is easier (in terms of deployment complexity) to perform theoretical experiments compared to Emulab. Also, the Emulab environment represents less risk compared to an OpenStack-based Cloud (all of OpenStack's intrinsic services architecture is very efficient but very complex, which includes higher risks of malfunction at any levels). In the experimentation we did, we did not noticed a big performance difference between the three environments except the network throughput and I/O speed variations on the Cloud-based experimental platform.

The security is an important parameter that mus be considered in any solution, especially the one deployed in a environment with high level of risk, like our case, where we have a huge number of users, and a large scale resources that compose the environment. This security parameter is considered by our framework MFHS, when the authentication in our system, the all messages exchanged between nodes, whatever their type (VM, PM, or other) are encrypted. The method that we have used is an asymmetric encryption method, also called public-key cryptography. This method is implemented in SSH protocol that we have re-used in MFHS. Also, this method is most suitable for large scale systems, because it reduces the number of keys exchanged.

5.5 Conclusion

This chapter was organized into two parts, the first was dedicated to the presentation of the evaluation of our proposed *InterRC* heuristic, whereas, the second one aimed to show the well-functioning of *MFHS* framework, meanwhile, showing the good results of our proposed heuristic *MMin*.

To achieve our objective, we have used our owner simulator to evaluate InterRC heuristic, and we have used three different environments for MMin evaluation and MFHS validation. The first of which is a simple environment that allows to analyze the scheduling algorithms only by simulation, whereas the second and the third ones are respectively, an emulator called *Emulab* and a real Cloud environment based on *OpenStack*. Note that the portability property of our framework is used during the transition from an environment to another one. During our experiments, we have explored all presented modules, and then quantified all considered QoS parameters.

Conclusion and Perspectives

In this last stage of our work presentation, we will give a general conclusion, and perspectives for future work. We begin by presenting a general conclusion that concerns all the works presented in our thesis. We then present the perspectives part as a set of open research challenges related to our works. We expect that tackling those challenges will contritbue to further advance the area.

Conclusion

In our thesis, we have mainly addressed two problems. The first one is related to task scheduling, whereas the second one is related to resource management. In both cases, we have focused on large scale distributed computing architectures.

The complexity of task scheduling problem lead the researchers working in this area to propose a huge number of approaches. These approaches, although, they often produce good results, are almost always only evaluated through by simulation and are therefore unfortunately never deployed and tested in real conditions.

Looking at the variety in terms of the different distributed computing architectures existing nowadays on one side, and the huge number of task scheduling approaches already proposed in the literature on the other side, it becomes more difficult to say which task scheduling approach is the more suitable for which distributed architectures, especially when it comes to real world environments. To help resolve this issue, it is important to design a smart resource management solution that can: (1) be adapted to any distributed computing architecture, and (2) help both the research and development communities to make an automatic switch from simulation to real deployment for any task scheduling algorithms in any distributed computing environment.

Our efforts to resolve the problems addressed in our thesis resulted in the following three major contributions:

The first one consists in proposing a smart resource management solution called

MFHS, which can be adapted to any distributed computing architecture, and that can help the research and development communities alike to achieve an important property that we have called "From theoretical to real deployment". The latter consists to ensure an automatic switch from the simulation phase to the real deployment phase of any task scheduling algorithms in any distributed computing architecture.

MFHS is a generic framework which includes dedicated modules in charge of discovering computing resources, collecting data about requests, scheduling (and allocating) tasks over the available resources, monitoring those resources, and analyzing the results through a set of parameters. As it presented in Chapter III the architecture of *MFHS* allows conducting experimentation from scratch in both virtual and real distributed computing environments for any kind of studies. Indeed, thanks to the *Resource Discovery* module, the setting up of experimentation in a heterogeneous environment is facilitated, since *MFHS* is able to automatically discover the main computing resources characteristics: number of Vcpu, amount of RAM, free storage capacity, read/write disk rate and upload/download rate between computing nodes. This functionality is one of the keys to ease the setting up of experiments in a real distributed computing context.

The second and the third contributions can be seen from a pure scheduling point of view. Both proposed methods, *MMin* and *InterRC*, aim to optimize the total response time.

InterRC is an evolutionary heuristic. In general, evolutionary heuristics are not deterministic, which is also the case of *InterRC*; this means that the results may not be the same if we re-execute the scheduler. *InterRC* uses a new concept that we have proposed, called Inter Collaboration, which allows a set of resources to work together by trying to exchange tasks with the aim to improve the total response time.

MMin is a fast deterministic scheduling heuristic, characterized by low execution time. *MMin* is proposed with the aim to get a better optimization trade-off while taking advantage of the good qualities of both well-known *Min-Min* and *Max-Min* heuristics. Like the latter two heuristics, *MMin* is executed in iterations, in each iteration, *MMin* call *Min-Min* or *Max-Min* according to the global state of the system.

In addition to the proposition of two scheduling heuristics and a resource management solution, we have presented, in this thesis, two chapters that describe the state of the art. The first one discusses the background of the proposed research of this thesis, and the second one describes the work in relation to our thesis work.

The experimentation shown the throughout Chapter V illustrates how the *MFHS* framework can be easily integrated into various distributed computing environments.

This is achieved using two different environments which are a real Cloud environment based on *OpenStack* and a test-bed environment based on *Emulab*. Through the set of experimentations, we have demonstrated the possibility to easily integrate new task scheduling algorithms into the proposed framework. Thus, the framework allows selecting some or all of the available algorithms, in order to make a theoretical evaluation before conducting a real experimentation. Another interesting point offered by the framework, is the ability to compute a coefficient (called Pi) which makes it possible to predict the percentage of offset expected between the theoretical results and those that should be obtained after a real execution.

Also, the set of conducted experimentations have shown the intrinsic ability of *MFHS* to compute and then allowed us to compare scheduling approaches through various metrics. These metrics can be either multiple objectives to be optimized or simple parameters to be analyzed. While the *MMin* heuristic used in experimentation only focuses on its optimization using the makespan metric, all the other analysed results include other parameters like the average resource utilization, the cost, and the global energy consumption.

The evaluation of *InterRC* was conducted with a set of non evolutionary heuristics, then with a set of evolutionary heuristics. In the first case, *InterRC* always gives the best total completion time, whereas it gives the best total completion time in about 90% of the experiùe,ts in the second case. A weak point of *InterRC*, and most of the other evolutionary heuristics, is their execution time, which is usually high compared with non-evolutionary heuristics. In addition, the results can stop improving after some time, making it important to detect the point of "evolution stop" to interrupt the algorithm execution.

Perspectives

Despite substantial contributions of the current thesis in the resources management and scheduling in large distributed systems, there are several perspectives that can follows the presented work. We will present a set of the perspectives in three parts, each one discussing the perspectives related to a corresponding contribution:

InterRC algorithm : Since the *InterRC* heuristic proposes to the different resources to collaborate together in order to improve the total response time, and knowing that any resource of the whole set of resources can crash in any time, it will be interesting to think about the integration of the fault tolerance management

aspect in order to allow the *InterRC* heuristic to continue its execution if one or more resources failed to continue their work.

It is also possible to make more intensive experimentations by varying the time dedicated to the *InterRC* execution and show the corresponding results. It would finally be useful to allow more variety in the initial solutions and show the behavior of *InterRC* in regard to the initial solution.

MMin algorithm Currently, *MMin* combines only *Max-Min* and *Min-Min* heuristics. It is conceivable to combine other heuristics, including evolutionary ones, comparing the resulting combination with other scheduling algorithms. Integrating more algorithms into the *MFHS* framework would increase its chances of being used by others researches and developers.

Many common perspectives can concern both *InterRC* and *MMin* heuristics:

- Both *InterRC* and *MMin* heuristics are mono-objective, their aim being to minimize the total response time optimization. It is conceivable to converting them into multi-objective heuristics, by introducing other objectives to be optimized, especially cost and energy consumption. We could then compare them with multi-objective scheduling algorithms.
- Considering the tasks' characteristics, both *InterRC* and *MMin* are oriented towards scheduling non-preemptive and independent tasks with equal properties. It is possible to extend our algorithms to schedule tasks with different nature, like dependent tasks that form a DAG, tasks with unequal priorities, or preemptive tasks, ...
- it is conceivable to combine *InterRC*, *MMin*, or their inherited algorithms with others heuristics while using our *MFHS* framework. That would allow us to both get more available algorithms in *MFHS*, and get more comparison results concerning these scheduling algorithms.

MFHS Framework The future works that we propose as improvements to our framework can be summarized in the following points:

- Extend the framework through other specific features closer to actual emerging distributed and heterogeneous architecture like Fog Computing,
- Extend *MFHS* to support other nature of tasks, inter alia, the tasks that can be associated to a type which defines its priority or to tasks that can be delayed or stopped (pre-emption) for a certain period of time,

- Extend *MFHS* to support other nature of schedulers, like event-invocation, and dynamic scheduling.
- Integrate a fault tolerance management mechanism, in order to ensure the continuity of the execution of such scheduling algorithms in the case of fault detection,
- Perform a detailed analysis of the scalability of both *MFHS* modules and the *MMin* heuristic to ensure the proper functioning of *MFHS* in a wide Internet of Things environment,
- Conduct experimentations in an experimental distributed computing plateform which is partially feeded by renewable energy (photovoltaic energy production). This would allow to improve the *MFHS* framework by taking into account both brown and green energy sources. The analysis of the global amount of consumed energy could be done while taking into account the productivity rate of the various energy sources with the intention of increasing the use of clean energy and improving the way they are used, given their limited lifetime.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Mohammad Aazam and Eui-Nam Huh. Fog computing and smart gateway based communication for cloud of things. In *2014 International Conference on Future Internet of Things and Cloud*, pages 464–470. IEEE, 2014.

[2] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and S. Ali. Task execution time modeling for heterogeneous computing systems. In *Proceedings 9th Heterogeneous Computing Workshop (HCW 2000) (Cat. No.PR00556)*, pages 185–199, 2000. doi: 10.1109/HCW.2000.843743.

[3] Slawomir Bak, Marcin Krystek, Krzysztof Kurowski, Ariel Oleksiak, Wojciech Piatek, and Jan Waglarz. Gssim-a tool for distributed computing experiments. *Scientific Programming*, 19(4):231–251, 2011.

[4] Rajesh Kumar Bawa and Gaurav Sharma. Modified min-min heuristic for job scheduling based on qos in grid environment. In *Information Management in the Knowledge Economy (IMKE), 2013 2nd International Conference on*, pages 166–171. IEEE, 2013.

[5] AS Ajeena Beegom and MS Rajasree. A particle swarm optimization based pareto optimal task scheduling in cloud computing. In *International Conference in Swarm Intelligence*, pages 79–86. Springer, 2014.

[6] Anton Beloglazov and Rajkumar Buyya. Openstack neat: a framework for dynamic and energy-efficient consolidation of virtual machines in openstack clouds. *Concurrency and Computation: Practice and Experience*, 27(5):1310–1333, 2015.

[7] Upendra Bhoi, Purvi N Ramanuj, et al. Enhanced max-min task scheduling algorithm in cloud computing. *International Journal of Application or Innovation in Engineering and Management (IJAIEM)*, 2(4):259–264, 2013.

[8] Supriya S. Bichkule and Ravi Mante. Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds : a review. 2017.

[9] Luiz F Bittencourt, Javier Diaz-Montes, Rajkumar Buyya, Omer F Rana, and Manish Parashar. Mobility-aware application scheduling in fog computing. *IEEE Cloud Computing*, 4(2):26–35, 2017.

[10] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16, 2012.

[11] Flavio Bonomi, Rodolfo Milito, Preethi Natarajan, and Jiang Zhu. Fog computing: A platform for internet of things and analytics. In *Big data and internet of things: A roadmap for smart environments*, pages 169–186. Springer, 2014.

[12] Tracy D Braun, Howard Jay Siegel, Noah Beck, Ladislau L Bölöni, Muthucumaru Maheswaran, Albert I Reuther, James P Robertson, Mitchell D Theys, Bin Yao, Debra Hensgen, and Richard F Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810 – 837, 2001. ISSN 0743-7315. doi: https://doi.org/10.1006/jpdc.2000.1714. URL `http://www.sciencedirect.com/science/article/pii/S0743731500917143`.

[13] Marc Bux and Ulf Leser. Dynamiccloudsim: Simulating heterogeneity in computational clouds. *Future Generation Computer Systems*, 46:85–99, 2015.

[14] Charles C Byers and Patrick Wetterwald. Fog computing distributing data and intelligence for resiliency and scale necessary for iot: The internet of things (ubiquity symposium). *Ubiquity*, 2015(November):1–12, 2015.

[15] Zhicheng Cai, Qianmu Li, and Xiaoping Li. Elasticsim: A toolkit for simulating workflows with cloud resource runtime auto-scaling and stochastic task execution times. *Journal of Grid Computing*, 15(2):257–272, 2017.

[16] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1):23–50, 2011.

[17] Rodrigo N Calheiros, Marco AS Netto, César AF De Rose, and Rajkumar Buyya. Emusim: an integrated emulation and simulation environment for modeling, evaluation, and validation of performance of cloud computing applications. *Software: Practice and Experience*, 43(5):595–612, 2013.

[18] Jan Carlson, Jukka Mäki-Turja, and Mikael Nolin. Event-Pattern Triggered Real-Time Tasks. In Giorgio Buttazzo and Pascale Mine, editors, *16th International Conference on Real-Time and Network Systems (RTNS 2008)*, Rennes, France, Oct 2008.

[19] Henri Casanova, Arnaud Legrand, and Martin Quinson. Simgrid: A generic framework for large-scale distributed experiments. In *Computer Modeling and Simulation, 2008. UKSIM 2008. Tenth International Conference on*, pages 126–131, april 2008.

[20] H. J. Chang, J. J. Wu, and P. Liu. Job scheduling techniques for distributed systems with heterogeneous processor cardinality. In *2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks*, pages 57–62, Dec 2009. doi: 10.1109/I-SPAN.2009.68.

[21] Huankai Chen, F. Wang, N. Helian, and G. Akanmu. User-priority guided min-min scheduling algorithm for load balancing in cloud computing. In *2013 National Conference on Parallel Computing Technologies (PARCOMPTECH)*, pages 1–8, Feb 2013. doi: 10.1109/ParCompTech.2013.6621389.

[22] W. N. Chen and J. Zhang. A set-based discrete pso for cloud workflow scheduling with user-defined qos constraints. In *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 773–778, Oct 2012. doi: 10.1109/ICSMC.2012.6377821.

[23] Weiwei Chen and Ewa Deelman. Workflowsim: A toolkit for simulating scientific workflows in distributed environments. In *E-science (e-science), 2012 IEEE 8th International Conference on*, pages 1–8. IEEE, 2012.

[24] S Chitra, B Madhusudhanan, GR Sakthidharan, and P Saravanan. Local minima jump pso for workflow scheduling in cloud computing environments. In *Advances in computer science and its applications*, pages 1225–1234. Springer, 2014.

[25] Santanu Dam, Gopa Mandal, Kousik Dasgupta, and Paramartha Dutta. An ant colony based load balancing strategy in cloud computing. In *Advanced Computing, Networking and Informatics-Volume 2*, pages 403–413. Springer, 2014.

[26] Marco Dorigo and Gianni Di Caro. Ant colony optimization: a new meta-heuristic. In *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*, volume 2, pages 1470–1477. IEEE, 1999.

[27] Rubing Duan, Radu Prodan, and Thomas Fahringer. Performance and cost optimization for multiple large-scale grid workflow applications. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, page 12. ACM, 2007.

[28] OM Elzeki, MZ Reshad, and MA Elsoud. Improved max-min algorithm in cloud computing. *International Journal of Computer Applications*, 50(12), 2012.

[29] Larry J. Eshelman. The chc adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. volume 1 of *Foundations of Genetic Algorithms*, pages 265 – 283. Elsevier, 1991. doi: https://doi.org/10.1016/B978-0-08-050684-5.50020-3. URL `http://www.sciencedirect.com/science/article/pii/B9780080506845500203`.

[30] Kobra Etminani and M Naghibzadeh. A min-min max-min selective algorihtm for grid task scheduling. In *2007 3rd IEEE/IFIP International Conference in Central Asia on Internet*, pages 1–7. IEEE, 2007.

[31] Kobra Etminani, Mahmaud Naghibzadeh, and Noorali Raeeji Yanehsari. A hybrid min-min max-min algorithm with improved performance. *Department of Computer Engineering, Ferdowsi University of Mashad, Iran*, 32:1–3, 2007.

[32] Eugen Feller, Louis Rilling, and Christine Morin. Snooze: A scalable and autonomic virtual machine management framework for private clouds. In *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 482–489. IEEE, 2012.

[33] Mohamed Firdhous, Osman Ghazali, and Suhaidi Hassan. Fog computing: Will it be the future of cloud computing? The Third International Conference on Informatics & Applications (ICIA2014), 2014.

[34] Peter Xiang Gao, Andrew R Curtis, Bernard Wong, and Srinivasan Keshav. It's not easy being green. *ACM SIGCOMM Computer Communication Review*, 42(4):211–222, 2012.

[35] Pedro Garcia Lopez, Alberto Montresor, Dick Epema, Anwitaman Datta, Teruo Higashino, Adriana Iamnitchi, Marinho Barcellos, Pascal Felber, and Etienne Riviere. Edge-centric computing: Vision and challenges, 2015.

[36] Saurabh Kumar Garg and Rajkumar Buyya. Networkcloudsim: Modelling parallel applications in cloud simulations. In *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, pages 105–113. IEEE, 2011.

[37] Michael R Gary and David S Johnson. Computers and intractability: A guide to the theory of np-completeness, 1979.

[38] Y. Ge and G. Wei. Ga-based task scheduler for the cloud computing systems. In *2010 International Conference on Web Information Systems and Mining*, volume 2, pages 181–186, Oct 2010. doi: 10.1109/WISM.2010.87.

[39] Arash Ghorbannia Delavar and Yalda Aryan. Hsga: a hybrid heuristic algorithm for workflow scheduling in cloud systems. *Cluster Computing*, 17(1): 129–137, Mar 2014. ISSN 1573-7543. doi: 10.1007/s10586-013-0275-6. URL `https://doi.org/10.1007/s10586-013-0275-6`.

[40] Tarun Kumar Ghosh, Rajmohan Goswami, Sumit Bera, and Subhabrata Barman. Load balanced static grid scheduling using max-min heuristic. In *Parallel Distributed and Grid Computing (PDGC), 2012 2nd IEEE International Conference on*, pages 419–423. IEEE, 2012.

[41] Christos Gogos, Christos Valouxis, Panayiotis Alefragis, George Goulas, Nikolaos Voros, and Efthymios Housos. Scheduling independent tasks on heterogeneous processors using heuristics and column pricing. *Future Generation Computer Systems*, 60:48 – 66, 2016. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2016.01.016. URL `http://www.sciencedirect.com/science/article/pii/S0167739X16000297`.

[42] David E Golberg. Genetic algorithms in search, optimization, and machine learning. *Addion wesley*, 1989(102):36, 1989.

[43] S. Goutam and A. K. Yadav. Preemptable priority based dynamic resource allocation in cloud computing with fault tolerance. In *2015 International Conference on Communication Networks (ICCN)*, pages 278–285, Nov 2015. doi: 10.1109/ICCN.2015.54.

[44] Lizheng Guo, Shuguang Zhao, Shigen Shen, and Changyuan Jiang. Task scheduling optimization in cloud computing based on heuristic algorithm. *JNW*, 7:547–553, 2012.

[45] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K Ghosh, and Rajkumar Buyya. ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. *Software: Practice and Experience*, 47(9):1275–1296, 2017.

[46] A. A. Haruna, L. T. Jung, and N. Zakaria. Design and development of hybrid integrated thermal aware job scheduling on computational grid environment. In *2015 International Symposium on Mathematical Sciences and Computing Research (iSMSC)*, pages 13–17, May 2015. doi: 10.1109/ISMSC.2015.7594020.

[47] Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia Lawall. Entropy: a consolidation manager for clusters. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 41–50. ACM, 2009.

[48] Cristian Hernandez Benet, Robayet Nasim, Kyoomars Alizadeh Noghani, and Andreas Kassler. Openstackemu-a cloud testbed combining network emulation with openstack and sdn. In *The 14th Annual IEEE Consumer Communications & Networking Conference (CCNC), 8-11 Jan. 2017, Las Vegas, USA*. IEEE conference proceedings, 2017.

[49] Mike Hibler, Robert Ricci, Leigh Stoller, Jonathon Duerig, Shashi Guruprasad, Tim Stack, Kirk Webb, and Jay Lepreau. Large-scale virtualization in the emulab network testbed. In *USENIX 2008 Annual Technical Conference*, ATC'08, pages 113–128, 2008.

[50] Tran Cong Hung, Le Ngoc Hieu, Phan Thanh Hy, and Nguyen Xuan Phi. Mmsia: Improved max-min scheduling algorithm for load balancing on cloud computing. In *Proceedings of the 3rd International Conference on Machine Learning and Soft Computing*, pages 60–64. ACM, 2019.

[51] Felix Hupfeld, Toni Cortes, Björn Kolbeck, Jan Stender, Erich Focht, Matthias Hess, Jesus Malo, Jonathan Marti, and Eugenio Cesario. The xtreemfs architecture—a case for object-based file systems in grids. *Concurrency and computation: Practice and experience*, 20(17):2049–2060, 2008.

[52] T. Issariyakul and E. Hossain. *Introduction to network simulator NS2*. Springer, 2011.

[53] Hesam Izakian, Ajith Abraham, and Václav Snasel. Comparison of heuristics for scheduling independent tasks on heterogeneous distributed environments. In *2009 International Joint Conference on Computational Sciences and Optimization*, volume 1, pages 8–12. IEEE, 2009.

[54] A. Jain and R. Kumar. A multi stage load balancing technique for cloud environment. In *2016 International Conference on Information Communication and Embedded Systems (ICICES)*, pages 1–7, Feb 2016. doi: 10.1109/ICICES.2016.7518921.

[55] Fatemeh Jalali, Kerry Hinton, Robert Ayre, Tansu Alpcan, and Rodney S Tucker. Fog computing may help to save energy in cloud computing. *IEEE Journal on Selected Areas in Communications*, 34(5):1728–1739, 2016.

[56] Yaser Jararweh, Zakarea Alshara, Moath Jarrah, Mazen Kharbutli, and Mohammad N Alsaleh. Teachcloud: a cloud computing educational toolkit. *International Journal of Cloud Computing 1*, 2(2-3):237–257, 2013.

[57] Yaser Jararweh, Fadi Ababneh Lo'ai Tawalbeh, Fadi Ababneh, Abdallah Khreishah, and Fahd Dosari. Scalable cloudlet-based mobile computing model. In *FNC/MobiSPC*, pages 434–441, 2014.

[58] Navdeep Kaur and Khushdeep Kaur. Improved max-min scheduling algorithm. *IOSR Journal of Computer Engineering (IOSR-JCE)*, 17(3):42–49, 2015.

[59] Rajwinder Kaur and Prasenjit Kumar Patra. Resource allocation with improved min-min algorithm. *International Journal of Computer Applications*, 76(15): 61–67, August 2013. Full text available.

[60] Rajwinder Kaur and Prasenjit Kumar Patra. Resource allocation with improved minmin algorithm. *International Journal of Computer Applications*, 76(15), 2013.

[61] Abdelhamid Khiat and Abdelkamel Tari. Interrc: An inter-resources collaboration heuristic for scheduling independent tasks on heterogeneous distributed environments. In *MENDEL*, volume 25, pages 179–188, 2019.

[62] Abdelhamid Khiat, Abdelkamel Tari, and Tom Guérout. Mfhs: A modular scheduling framework for heterogeneous system. *Software: Practice and Experience*, 50(8):1463–1497, 2020. doi: 10.1002/spe.2827. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2827`.

[63] Hyunjoo Kim and Manish Parashar. Cometcloud: An autonomic cloud engine. *Cloud Computing: Principles and Paradigms*, pages 275–297, 2011.

[64] Alexander Klemm, Christoph Lindemann, and Oliver P Waldhorst. A special-purpose peer-to-peer file sharing system for mobile ad hoc networks. In *2003 IEEE 58th Vehicular Technology Conference. VTC 2003-Fall (IEEE Cat. No. 03CH37484)*, volume 4, pages 2758–2763. IEEE, 2003.

[65] D. Kliazovich, P. Bouvry, Y. Audzevich, and S.U. Khan. Greencloud: A packet-level simulator of energy-aware cloud computing data centers. In *GLOBECOM 2010, IEEE Global Telecommunications Conference*, pages 1–5, 2010.

[66] Dzmitry Kliazovich, Pascal Bouvry, and Samee Ullah Khan. Greencloud: a packet-level simulator of energy-aware cloud computing data centers. *The Journal of Supercomputing*, 62(3):1263–1283, 2012.

[67] Andreas Kohne, Marc Spohr, Lars Nagel, and Olaf Spinczyk. Federated-cloudsim: a sla-aware federated cloud simulation framework. In *Proceedings of the 2nd International Workshop on CrossCloud Systems*, page 3. ACM, 2014.

[68] James Kok Konjaang, Fahrul Hakim Ayob, and Abdullah Muhammed. Cost effective expa-max-min scientific workflow allocation and load balancing strategy in cloud computing. *JCS*, 14(5):623–638, 2018.

[69] H. Kopetz. The time-triggered model of computation. In *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No.98CB36279)*, pages 168–177, Dec 1998. doi: 10.1109/REAL.1998.739743.

[70] Krzysztof Kurowski, Ariel Oleksiak, W Piatek, Tomasz Piontek, A Przybyszewski, and J Weglarz. Dcworms–a tool for simulation of energy efficiency in distributed computing infrastructures. *Simulation Modelling Practice and Theory*, 39:135–151, 2013.

[71] Adrien Lebre, Jonathan Pastor, Marin Bertier, Frédéric Desprez, Jonathan Rouzaud-Cornabas, Cédric Tedeschi, Paolo Anedda, Gianluigi Zanetti, Ramon Nou, Toni Cortes, et al. Beyond the cloud, how should next generation utility computing infrastructures be designed? 2013.

[72] Kun Li, Gaochao Xu, Guangyu Zhao, Yushuang Dong, and Dan Wang. Cloud task scheduling based on load balancing ant colony optimization. In *2011 sixth annual ChinaGrid conference*, pages 3–9. IEEE, 2011.

[73] Yan Li, Zhunge Zhu, and Yong Wang. Min-max-min: A heuristic scheduling algorithm for jobs across geo-distributed datacenters. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1573–1574. IEEE, 2018.

[74] Seung-Hwan Lim, Bikash Sharma, Gunwoo Nam, Eun-Kyoung Kim, and Chita R Das. Mdcsim: A multi-tier data center simulation, platform. In *CLUSTER*, volume 31, pages 1–9, 2009.

[75] Cong Liu and Sanjeev Baskiyar. A general distributed scalable grid scheduler for independent tasks. *Journal of Parallel and Distributed Computing*, 69(3): 307–314, 2009.

[76] X. Lu and Z. Gu. A load-adapative cloud resource scheduling model based on ant colony algorithm. In *2011 IEEE International Conference on Cloud Computing and Intelligence Systems*, pages 296–300, Sept 2011. doi: 10.1109/CCIS.2011.6045078.

[77] Muthucumaru Maheswaran, Shoukat Ali, Howard Jay Siegel, Debra Hensgen, and Richard F Freund. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *Journal of parallel and distributed computing*, 59(2):107–131, 1999.

[78] A. H. Mahmud and S. S. Iyengar. A distributed framework for carbon and cost aware geographical job scheduling in a hybrid data center infrastructure. In *2016 IEEE International Conference on Autonomic Computing (ICAC)*, pages 75–84, July 2016. doi: 10.1109/ICAC.2016.21.

[79] Francesco Malandrino, Scott Kirkpatrick, and Carla-Fabiana Chiasserini. How close to the edge? delay/utilization trends in mec. In *Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking*, pages 37–42, 2016.

[80] S. Mittal and A. Katal. An optimized task scheduling algorithm in cloud computing. In *2016 IEEE 6th International Conference on Advanced Computing (IACC)*, pages 197–202, Feb 2016. doi: 10.1109/IACC.2016.45.

[81] Shubham Mittal and Avita Katal. An optimized task scheduling algorithm in cloud computing. In *2016 IEEE 6th International Conference on Advanced Computing (IACC)*, pages 197–202. IEEE, 2016.

[82] Christine Morin, Pascal Gallard, Renaud Lottiaux, and Geoffroy Vallée. Towards an efficient single system image cluster operating system. *Future Generation Computer Systems*, 20(4):505–521, 2004.

[83] Sergio Nesmachnow, Héctor Cancela, and Enrique Alba. A parallel micro evolutionary algorithm for heterogeneous computing and grid scheduling. *Applied Soft Computing*, 12(2):626–639, 2012.

[84] Anant V Nimkar and Soumya K Ghosh. Realization of virtual resource management framework in iaas cloud federation. In *Proceedings of International Conference on Communication and Networks*, pages 155–164. Springer, 2017.

[85] Alberto Núñez, Javier Fernández, Rosa Filgueira, Félix García, and Jesús Carretero. Simcan: A flexible, scalable and expandable simulation platform for modelling and simulating distributed architectures and applications. *Simulation Modelling Practice and Theory*, 20(1):12–32, 2012.

[86] Alberto Núñez, Jose L Vázquez-Poletti, Agustin C Caminero, Gabriel G Castañé, Jesus Carretero, and Ignacio M Llorente. icancloud: A flexible and scalable cloud infrastructure simulator. *Journal of Grid Computing*, 10(1):185–209, 2012.

[87] Sanjaya Kumar Panda, Pratik Agrawal, Pabitra Mohan Khilar, and Durga Prasad Mohapatra. Skewness-based min-min max-min heuristic for grid task scheduling. *ACCT : Fourth International Conference on Advanced Computing & Communication Technologies ACCT*, 2014.

[88] Saeed Parsa and Reza Entezari-Maleki. Rasa: a new grid task scheduling algorithm. *International Journal of Digital Content Technology and its Applications*, 3(4):91–99, 2009.

[89] Frédéric Pinel, Bernabé Dorronsoro, and Pascal Bouvry. A new parallel asynchronous cellular genetic algorithm for scheduling in grids. In *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, pages 1–8. IEEE, 2010.

[90] Sareh Fotuhi Piraghaj, Amir Vahid Dastjerdi, Rodrigo N Calheiros, and Rajkumar Buyya. Containercloudsim: An environment for modeling and simulation of containers in cloud data centers. *Software: Practice and Experience*, 47(4): 505–521, 2017.

[91] Fahimeh Ramezani, Jie Lu, and Farookh Hussain. Task scheduling optimization in cloud computing applying multi-objective particle swarm optimization. In *International Conference on Service-oriented computing*, pages 237–251. Springer, 2013.

[92] A. Razaque, N. R. Vennapusa, N. Soni, G. S. Janapati, and k. R. Vangala. Task scheduling in cloud computing. In *2016 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, pages 1–5, April 2016. doi: 10.1109/LISAT.2016.7494149.

[93] Ubaid Ur Rehman, Amir Ali, and Zahid Anwar. seccloudsim: Secure cloud simulator. In *Frontiers of Information Technology (FIT), 2014 12th International Conference on*, pages 208–213. IEEE, 2014.

[94] B. P. Rimal and M. Maier. Workflow scheduling in multi-tenant cloud computing environments. *IEEE Transactions on Parallel and Distributed Systems*, 28 (1):290–304, Jan 2017. ISSN 1045-9219. doi: 10.1109/TPDS.2016.2556668.

[95] Suzanne Rivoire, Parthasarathy Ranganathan, and Christos Kozyrakis. A comparison of high-level full-system power models. In *Proceedings of the 2008 conference on Power aware computing and systems*, HotPower'08, pages 3–3, Berkeley, CA, USA, 2008. USENIX Association.

[96] Thiago Teixeira Sá, Rodrigo N Calheiros, and Danielo G Gomes. Cloudreports: an extensible simulation tool for energy-aware cloud computing environments. In *cloud computing*, pages 127–142. Springer, 2014.

[97] K. Sellami, M. Ahmed-Nacer, P.F. Tiako, and R. Chelouah. Immune genetic algorithm for scheduling service workflows with QoS constraints in cloud computing. *South African Journal of Industrial Engineering*, 24:68 – 82, 11 2013. ISSN 2224-7890.

[98] S. Selvarani and G. S. Sadhasivam. Improved cost-based algorithm for task scheduling in cloud computing. In *2010 IEEE International Conference on Computational Intelligence and Computing Research*, pages 1–5, Dec 2010. doi: 10.1109/ICCIC.2010.5705847.

[99] Neha Sharma, Sanjay Tyagi, and Swati Atri. A comparative analysis of min-min and max-min algorithms based on the makespan parameter. *International Journal of Advanced Research in Computer Science*, 8(3), 2017.

[100] Prateek Sharma, Tian Guo, Xin He, David Irwin, and Prashant Shenoy. Flint: Batch-interactive data-intensive processing on transient servers. In *Proceedings of the Eleventh European Conference on Computer Systems*, pages 1–15, 2016.

[101] Gang Shen and Yan-Qing Zhang. A shadow price guided genetic algorithm for energy aware task scheduling on cloud computers. In *International Conference in Swarm Intelligence*, pages 522–529. Springer, 2011.

[102] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016.

[103] Yuhui Shi et al. Particle swarm optimization: developments, applications and resources. In *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546)*, volume 1, pages 81–86. IEEE, 2001.

[104] Mohammad Shojafar, Saeed Javanmardi, Saeid Abolfazli, and Nicola Cordeschi. Fuge: A joint meta-heuristic approach to cloud job scheduling algorithm using fuzzy theory and a genetic method. *Cluster Computing*, 18(2): 829–844, June 2015. ISSN 1386-7857. doi: 10.1007/s10586-014-0420-x. URL `http://dx.doi.org/10.1007/s10586-014-0420-x`.

[105] S Sindhu and Saswati Mukherjee. Efficient task scheduling algorithms for cloud computing environment. In *International Conference on High Performance Architecture and Grid Computing*, pages 79–83. Springer, 2011.

[106] Thamarai Selvi Somasundaram and Kannan Govindarajan. Cloudrb: A framework for scheduling and managing high-performance computing (hpc) applications in science cloud. *Future Generation Computer Systems*, 34:47–65, 2014.

[107] John A. Stankovic, Krithi Ramamritham, and Marco Spuri. *Deadline Scheduling for Real-Time Systems: Edf and Related Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 1998. ISBN 0792382692.

[108] Ivan Stojmenovic and Sheng Wen. The fog computing paradigm: Scenarios and security issues. In *2014 federated conference on computer science and information systems*, pages 1–8. IEEE, 2014.

[109] E Kartal Tabak, B Barla Cambazoglu, and Cevdet Aykanat. Improving the performance of independenttask assignment heuristics minmin, maxmin and sufferage. *IEEE Transactions on Parallel and Distributed Systems*, 25(5):1244–1256, 2013.

[110] Andrew S Tanenbaum and Maarten Van Steen. *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[111] Fei Tao, Ying Feng, Lin Zhang, and T.W. Liao. Clps-ga: A case library and pareto solution-based hybrid genetic algorithm for energy-aware cloud service scheduling. *Applied Soft Computing*, 19(Supplement C):264 – 279, 2014. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2014.01.036. URL `http://www.sciencedirect.com/science/article/pii/S1568494614000568`.

[112] J. Teixeira, G. Antichi, D. Adami, A. Del Chiaro, S. Giordano, and A. Santos. Datacenter in a box: Test your sdn cloud-datacenter controller at home. In *2013 Second European Workshop on Software Defined Networks*, pages 99–104, Oct 2013. doi: 10.1109/EWSDN.2013.23.

[113] Lizhe Wang, Samee U. Khan, Dan Chen, Joanna Kołodziej, Rajiv Ranjan, Cheng zhong Xu, and Albert Zomaya. Energy-aware parallel task scheduling in a cluster. *Future Generation Computer Systems*, 29(7):1661 – 1670, 2013. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2013.02.010. URL `http://www.sciencedirect.com/science/article/pii/S0167739X13000484`. Including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services & Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond.

[114] Xiaofeng Wang, Chee Shin Yeo, Rajkumar Buyya, and Jinshu Su. Optimizing the makespan and reliability for workflow applications with reputation and a look-ahead genetic algorithm. *Future Generation Computer Systems*, 27(8): 1124–1134, 2011.

[115] Yan Wang, Jinkuan Wang, Cuirong Wang, and Xin Song. Research on resource scheduling of cloud based on improved particle swarm optimization algorithm. In *International Conference on Brain Inspired Cognitive Systems*, pages 118–125. Springer, 2013.

[116] Xianglin Wei, Jianhua Fan, Tongxiang Wang, and Qiping Wang. Efficient application scheduling in mobile cloud computing based on max–min ant system. *Soft Computing*, 20(7):2611–2625, 2016.

[117] X. Wen, M. Huang, and J. Shi. Study on resources scheduling based on aco allgorithm and pso algorithm in cloud computing. In *2012 11th International Symposium on Distributed Computing and Applications to Business, Engineering Science*, pages 219–222, Oct 2012. doi: 10.1109/DCABES.2012.63.

[118] Tom White. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[119] Bhathiya Wickremasinghe, Rodrigo N Calheiros, and Rajkumar Buyya. Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 446–452. IEEE, 2010.

[120] Timothy Wood, Prashant J Shenoy, Arun Venkataramani, Mazin S Yousif, et al. Black-box and gray-box strategies for virtual machine migration. In *NSDI*, volume 7, pages 17–17, 2007.

[121] Guanlin Wu, Weidong Bao, Xiaomin Zhu, and Xiongtao Zhang. A general cross-layer cloud scheduling framework for multiple iot computer tasks. *Sensors*, 18 (6):1671, 2018.

[122] Min-You Wu and Wei Shu. A high-performance mapping algorithm for heterogeneous computing systems. In *Proceedings 15th International Parallel and Distributed Processing Symposium. IPDPS 2001*, pages 6–pp. IEEE, 2001.

[123] Z. Wu, Z. Ni, L. Gu, and X. Liu. A revised discrete particle swarm optimization for cloud workflow scheduling. In *2010 International Conference on Computational Intelligence and Security*, pages 184–188, Dec 2010. doi: 10.1109/CIS.2010.46.

[124] Fatos Xhafa, Enrique Alba, Bernabé Dorronsoro, Bernat Duran, and Ajith Abraham. Efficient batch job scheduling in grids using cellular memetic algorithms. In *Metaheuristics for Scheduling in Distributed Computing Environments*, pages 273–299. Springer, 2008.

[125] Y. Xiaoguang, C. Tingbin, and Z. Qisong. Research on cloud computing schedule based on improved hybrid pso. In *Proceedings of 2013 3rd International Conference on Computer Science and Network Technology*, pages 388–391, Oct 2013. doi: 10.1109/ICCSNT.2013.6967136.

[126] Marcelo Yannuzzi, Rodolfo Milito, René Serral-Gracià, Diego Montero, and Mario Nemirovsky. Key ingredients in an iot recipe: Fog computing, cloud computing, and more fog computing. In *2014 IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pages 325–329. IEEE, 2014.

[127] Shanhe Yi, Zijiang Hao, Zhengrui Qin, and Qun Li. Fog computing: Platform and applications. In *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, pages 73–78. IEEE, 2015.

[128] Shanhe Yi, Zhengrui Qin, and Qun Li. Security and privacy issues of fog computing: A survey. In *International conference on wireless algorithms, systems, and applications*, pages 685–695. Springer, 2015.

[129] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, Ion Stoica, et al. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10): 95, 2010.

[130] Ben Zhang, Nitesh Mor, John Kolb, Douglas S Chan, Ken Lutz, Eric Allman, John Wawrzynek, Edward Lee, and John Kubiatowicz. The cloud is not enough: Saving iot from the cloud. In *7th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, 2015.

[131] C. Zhao, S. Zhang, Q. Liu, J. Xie, and J. Hu. Independent tasks scheduling based on genetic algorithm in cloud computing. In *2009 5th International Conference on Wireless Communications, Networking and Mobile Computing*, pages 1–4, Sept 2009. doi: 10.1109/WICOM.2009.5301850.

[132] Zhongni Zheng, Rui Wang, Hai Zhong, and Xuejie Zhang. An approach for cloud resource scheduling based on parallel genetic algorithm. In *2011 3rd International Conference on Computer Research and Development*, volume 2, pages 444–447, March 2011. doi: 10.1109/ICCRD.2011.5764170.

[133] Ao Zhou, Shangguang Wang, Chenchen Yang, Lei Sun, Qibo Sun, and Fangchun Yang. Ftcloudsim: support for cloud service reliability enhancement simulation. *International Journal of Web and Grid Services*, 11(4):347–361, 2015.

[134] Kai Zhu, Huaguang Song, Lijing Liu, Jinzhu Gao, and Guojian Cheng. Hybrid genetic algorithm for cloud computing applications. *2011 IEEE Asia-Pacific Services Computing Conference*, pages 182–187, 2011.

[135] Xiaomin Zhu, Yabing Zha, Ling Liu, and Peng Jiao. General framework for task scheduling and resource provisioning in cloud computing systems. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 664–673. IEEE, 2016.

[136] X. Zuo, G. Zhang, and W. Tan. Self-adaptive learning pso-based deadline constrained task scheduling for hybrid iaas cloud. *IEEE Transactions on Automation Science and Engineering*, 11(2):564–573, April 2014. ISSN 1545-5955. doi: 10.1109/TASE.2013.2272758.

ABSTRACT

In today's distributed systems, heterogeneous tasks appear concurrently at any site and at any time. However, Task scheduling presents a major problem in distributed systems. Indeed, it is difficult to: (i) know the location of all tasks/resources, (ii) verify/guarantee the time constraints taking into account the communication delays induced by the tasks, and (iii) predict the platform's behavior. In this thesis, we have contributed to the resolution of these problems by proposing task scheduling strategies oriented to distributed systems. Also, by developing resource management solutions which allow, among others, resource discovery, task discovery, resource allocation, resource behavior prediction, etc...

Keywords: Distributed computing; Heterogeneous environment; Task scheduling; Resource management.

RÉSUMÉ

Dans les systèmes distribués, des tâches hétérogènes peuvent apparaître concurremment sur n'importe quel site et à n'importe quel moment. Le problème d'ordonnancement des tâches dans un univers distribué est mal résolu à l'heure actuelle. En effet, il est difficile de : i) connaître la localisation de toutes les tâches/ressources, ii) vérifier/garantir les contraintes temporelles en tenant compte des délais de communication induits par les tâches, et iii) prédire le comportement de la plate-forme. Dans cette thèse, nous avons contribué à la résolution de ces problèmes. Ceci en proposant des stratégies d'ordonnancement des tâches orientées vers les systèmes distribués. Ainsi, en élaborant des solutions de gestion de ressources qui permettent, entre autres, d'assurer la découverte des ressources, la découverte des tâches, l'allocation des ressources et la prédiction de comportement des ressources, etc ...

Mots-clés : Informatique distribuée; Environnement hétérogène; Ordonnancement des tâches; Gestion de ressources.

ملخص :

في أنظمة الحوسبة الموزعة ، يمكن أن تظهر مهام غير متجانسة بشكل متزامن في أي موقع وفي أي وقت . حتى يومنا هذا ، لم يتم التوصل إلى حل نهائي لمشكلة جدولة المهام في وسط موزع بالكامل. في الواقع ، من الصعب: (١) معرفة موقع جميع المهام / الموارد ، (٢) التحقق / ضمان قيود الوقت مع مراعاة تأخيرات الاتصال التي تسببها المهام ، و (٣) التنبؤ بسلوك النظام الموزع المستخدم قبل وبعد إستعمال خوارزمية جدولة المهام. لحل هذه المشاكل ، في هذه الأطروحة ، تم تنفيذ مجموعة من المساهمات الأصلية. حيث تم اقتراح استراتيجيات جديدة لجدولة المهام الموجهة للأنظمة الموزعة الحديثة ، بالإضافة أيضًا إلى تطوير حلول إدارة الموارد التي تسمح ، من بين أمور أخرى ، باكتشاف الموارد ، واكتشاف المهام ، وتخصيص الموارد ، والتنبؤ بسلوك الموارد ، إلخ.

الكلمات الدّالة : الحوسبة الموزعة؛ بيئة غير متجانسة ؛ جدولة المهام ؛ إدارة الموارد.