Multi-Constrained Scheduling Algorithms for Large-Scale Data Applications on Cloud Platforms

Résumé : L'ordonnancement des workflows dans les systèmes hétérogènes de type Cloud Computing est considéré comme un problème difficile. En conséquence, de nombreux algorithmes ne sont pas efficaces pour le traitement de données à grande échelle. En d'autres termes, les algorithmes traditionnels n'intègrent pas certains principes de base de Cloud Computing tels que l'élasticité et l'hétérogénéité des ressources informatiques. Par conséquent, dans les travaux de cette thèse, nous proposons trois stratégies d'ordonnancement et d'allocation des ressources. Premièrement, nous proposons une stratégie d'équilibrage de charge qui équilibre d'abord la charge entre les centres de traitement de données (datacenters), puis minimise le volume de données échangées. Deuxièmement, nous présentons un algorithme nommé DT-MG, qui vise à réduire la consommation d'énergie et à garantir les exigences de QoS (Quality of Services) en utilisant la théorie des jeux. Enfin, nous proposons une amélioration de l'algorithme HEFT (Heterogeneous Earliest Finish Time) sous la contrainte financière spécifiée par les utilisateurs pour bien équilibrer la charge des machines virtuelles et minimiser le temps d'exécution. Pour évaluer la performance de nos algorithmes, nous les avons comparés avec d'autres algorithmes d'ordonnancement des workflows en utilisant le simulateur Cloudsim. Les résultats rapportés montrent que nos algorithmes proposés ont de meilleures performances que les autres algorithmes en termes de tous les critères cités précédemment

Mots clés : Cloud Computing ; Big Data ; Workflows scientifiques ; Équilibrage de charge ; Consommation d'énergie ; Ordonnancement des tâches ; Tolérance aux fautes

---

Abstract: Scientific workflow applications have a complex structure and many discrete tasks; each task may include getting data, accessing software, or executing functions. For these reasons, the workflow scheduling in cloud computing is considered to be a difficult problem. As a result, over several years, many algorithms have been proposed to solve this problem. However, the majority of these traditional algorithms may not be efficient for processing large-scale data. In other words, traditional algorithms fail to incorporate some basic principles of cloud computing such as the elasticity and the heterogeneity of computing resources. As a consequence, this thesis proposes three scheduling algorithms that are capable of answering key issues to solve large-scale data scheduling problems. First, we propose a threshold-based load balancing algorithm, which first balances the load among datacentres, and afterwards minimizes the overhead of data exchanges. Second, we present an efficient algorithm named DT-MG, which aims to reduce the energy consumption and to guarantee the quality of service (QoS) requirements using matching game theory. Finally, we propose an enhancement of Heterogeneous Earliest Finish Time (HEFT) algorithm under a userspecified financial constraint to achieve a well-balanced load across the virtual machines as well as to minimize the workflow execution time. To evaluate the performance of our proposed algorithms, we compared them with the state-of-the-art workflow scheduling algorithms using CloudSim simulator. The reported results show that our proposed algorithms outperform existing scheduling algorithms in terms of all previously cited criteria

Keywords: Cloud computing; Big Data; Scientific workflows; Load balancing; Data placement; Energy consumption; Task scheduling; Fault tolerance.

Yassir SAMADI

Multi-Constrained Scheduling Algorithms for Large-Scale Data Applications on Cloud Platforms

Année : 2019 N° these : 136/ST2I

Année : 2019 **Thèse N° : 136/ST2I**

جامعة محمد الخامس بالرباط
Université Mohammed V de Rabat

École Nationale Supérieure d'Informatique et d'Analyse des systèmes
Centre d'Études Doctorales en Sciences des Technologies de l'Information et de l'Ingénieur

THÈSE DE DOCTORAT

MULTI-CONSTRAINED SCHEDULING ALGORITHMS FOR LARGE-SCALE DATA APPLICATIONS ON CLOUD PLATFORMS

Présentée par

Yassir SAMADI

Le 08/03/2019

Formation doctorale : Informatique
Structure de recherche : Smart System Laboratory

JURY

Professeur Mohamed ESSAAIDI PES, ENSIAS, Université Mohammed V de Rabat	**Président**
Professeur Mostapha ZBAKH PH, ENSIAS, Université Mohammed V de Rabat	**Directeur de thèse**
Professeur Claude TADONKI HDR, École des Mines de Paris	**Co-Encadrant de thèse**
Professeur Karim BAÏNA PES, ENSIAS, Université Mohammed V de Rabat	**Rapporteur**
Professeur Mohammed MEKNASSI PES, FSDM, Université Sidi Mohamed Ben Abdellah, Fès	**Rapporteur**
Professeur Christophe CÉRIN Professeur des universités, Université Paris 13, Paris	**Rapporteur**
Professeur Mostafa DAOUDI PES, FS, Université Mohammed Premier, Oujda	**Examinateur**

DEDICATION

Every challenging work needs self efforts as well as guidance of elders mainly those who are very close to our heart.
I dedicate this thesis to my sweet and loving parents and all my family.
You have successfully made me the person I am becoming.

ACKNOWLEDGEMENTS

The research work of this doctoral thesis has been carried out in the Smart System Laboratory (SSL), National School of Computer Science and System Analysis, University of Mohammed V, Rabat, Morocco and Centre de Recherche en Informatique (CRI), Mines ParisTech, Paris, France; During the years 2015-2018.

I am profoundly thankful to my advisor, Professor Mostapha Zbakh. I will always be indebted for all the support, guidance, encouragement, his friendly mood that makes it enjoyable to work with him, and understanding he has given me throughout these years. His endless patience and didactic attitude, combined with his wide knowledge, makes him a wonderful advisor. It is a great honor to work with you.

I am also indebted to my co-advisor Professor Claude Tadonki, who has been a constant source of encouragement, strong support and guidance during this thesis project. Thank you for your guidance that helped me continue my research in the right direction and the invaluable suggestions, and wonderful feedback. I am grateful for the hospitality you showed me during my stays that I performed in your laboratory, and also for always having the door open and willing to discuss and share you experience and wisdom to improve this dissertation.

I am most grateful to my reviewers Professor Karim BAÏNA, ENSIAS, Mohammed V University, Professor Mohammed MEKNASSI, Faculty of Sciences Dhar El Mahraz, and Professor Christophe CÉRIN, Paris 13 University for their thorough examination of the dissertation. Their detailed comments significantly improved the quality of this thesis.

A big thanks also to the entire CRI laboratory team, for their welcome that allowed me to work in a friendly environment, where mutual help and good mood.

Most importantly, my deepest gratitude also goes to my parents Mehdi Samadi and Rahma El mrini who have always filled my life with generous love, and unconditional support. This doctorate is truly theirs. My thanks go also to my sisters and brothers, especially my brother Elnouaman Samadi for his endless moral and financial supports throughout my career. To them I dedicate this thesis.

Yassir Samadi

Abstract

Cloud computing is a newly emerged computing platform that builds on the latest achievements of diverse research areas. It has been widely used due to its significant benefits and its ability to cope with large-scale data such as scientific workflows and big data applications. Scientific workflows describe a series of computations that enable the analysis of data in a structured and distributed manner. To run these workflows, the tasks must be scheduled on computational resources. The scheduling of a task consists of assigning it to a specific resource in order to fulfill a final goal such as minimizing the overall workflow execution time. Workflow applications have a complex structure and many discrete tasks; each task may include getting data, accessing software, or executing functions. For these reasons, the workflow scheduling is considered to be a difficult problem. Then, efficient scheduling algorithms are required for selection of the best suitable resources for workflow execution. As a result, over several years, many algorithms have been proposed to solve this problem. However, the majority of these traditional algorithms may not be efficient for processing large-scale data. In other words, traditional algorithms fail to incorporate some basic principles of cloud computing such as the elasticity and the heterogeneity of computing resources. Therefore, a proper solution of the workflow scheduling problem requires the analysis of tasks and resources, and, most important, the adaptation of optimization techniques. To achieve these goals, this thesis proposes three scheduling algorithms that are capable of answering key issues to solve large-scale data scheduling problem. First, we propose a threshold-based load balancing algorithm, which first balances the load among datacentres, and afterwards minimizes the overhead of data exchanges. Second, we present an efficient algorithm named DTMG, which aims to reduce the energy consumption and to guarantee the quality of service (QoS) requirements using *matching game theory*. Finally, we propose an enhancement of Heterogeneous Earliest Finish Time (HEFT) algorithm under a user-specified financial constraint to achieve a well-balanced load across the virtual machines as well as to minimize the workflow execution time. To evaluate the performance of our proposed algorithms, we compared them with the state-of-the-art workflow scheduling algorithms using Cloudsim simulator. The reported results show that our proposed algorithms outperform existing scheduling algorithms in terms of all previously cited criteria.

Keywords : Cloud computing; Big Data; Scientific workflows; Load balancing; Data placement; Energy consumption; Task scheduling; Fault tolerance.

Résumé

Le Cloud Computing est de plus en plus reconnu comme une nouvelle façon d'utiliser, à la demande, les services de calcul et de stockage de manière transparente et efficace. L'ordonnancement des *workflows*, notamment les *workflows* scientifiques, ainsi que l'allocation de ressources dans les systèmes hétérogènes de type Cloud Computing, suscitent une attention croissante avec l'augmentation de la popularité de Cloud Computing. Les *workflows* scientifiques ont une structure complexe et de nombreuses tâches discrètes; chaque tâche peut inclure la saisie de données, l'accès au logiciel, ou des fonctions de traitement. Pour ces raisons, l'ordonnancement du workflow est considérée comme un problème difficile. Ainsi, des algorithmes efficaces d'ordonnancement sont nécessaires pour la meilleure sélection des ressources. En conséquence, de nombreux algorithmes pour l'ordonnancement dans le cloud computing ont été proposés. Cependant, la plupart de ces algorithmes ne sont pas efficaces pour le traitement de données à grande échelle. En d'autres termes, les algorithmes traditionnels n'intègrent pas certains principes de base de cloud computing tels que l'élasticité et l'hétérogénéité des ressources informatiques. Par conséquent, une solution appropriée du problème d'ordonnancement des *workflows* nécessite l'analyse des tâches et des ressources, ainsi que l'adaptation des techniques d'optimisation. Dans les travaux de cette thèse, nous proposons trois stratégies d'ordonnancement et d'allocation des ressources. Premièrement, nous proposons une stratégie d'équilibrage de charge qui équilibre d'abord la charge entre les centres de traitement de données (datacenters), puis minimise le volume de données échangées. Deuxièmement, nous présentons un algorithme nommé DT-MG, qui vise à réduire la consommation d'énergie et à garantir les exigences de QoS (Quality of Services) en utilisant la *théorie des jeux*. Enfin, nous proposons une amélioration de l'algorithme HEFT (Heterogeneous Earliest Finish Time) sous la contrainte financière spécifiée par les utilisateurs pour bien équilibrer la charge des machines virtuelles et minimiser le temps d'exécution. Pour évaluer la performance de nos algorithmes, nous les avons comparés avec d'autres algorithmes d'ordonnancement des *workflows* en utilisant le simulateur Cloudsim. Les résultats rapportés montrent que nos algorithmes proposés ont de meilleures performances que les autres algorithmes en termes de tous les critères cités précédemment.

Mots-clés : Cloud computing; Big Data; *Workflows* scientifiques; Équilibrage de charge; Consommation d'énergie; Ordonnancement des tâches; Tolérance aux pannes.

Contents

List of Figures

List of Tables

List of Acronyms

ANN - **A**rtificial **N**eural **N**etwork
CSP - **C**loud **S**ervice **P**rovider
DAG - **D**irected **A**cyclic **G**raph
DI - **D**egree of **I**mbalance
DVFS - **D**ynamic **V**oltage and **F**requency **S**caling
EC2 - **E**lastic **C**ompute **C**loud
E-HEFT - **E**nhancement **H**eterogeneous **E**arliest **F**inish **T**ime
FARS - **F**ailure-**A**ware **R**esource **S**cheduling
FITS - **F**lexible **I**mage **T**ransport **S**ystem
GA - **G**enetic **A**lgorithm
GC - **G**lobal **C**ontroller
GFS - **G**oogle **F**ile **S**ystem
HDFS - **H**adoop **D**istributed **F**ile **S**ystem
HEFT - **H**eterogeneous **E**arliest **F**inish **T**ime
HPCC - **H**igh **P**erformance **C**omputing **C**hallenge
IaaS - **I**nfrastructure **A**s **A** **S**ervice
IWD - **I**ntelligent **W**ater **D**rops
LC - **L**ocal **C**ontroller
MadMmt - **M**edian **A**bsolute **D**eviation and **M**inimum **M**igration **T**ime
MadRs - **M**edian **A**bsolute **D**eviation and **R**andom **S**election
MBFD - **M**odified **B**est **F**it **D**ecreasing
MCFE - **M**onte **C**arlo **F**ailure **E**stimation
MET - **M**inimum **E**xecution **T**ime
MIPS - **M**illions of **I**nstructions **P**er **S**econd
MRBS - **M**ap**R**educe **B**enchmark **S**uite
NIST - **N**ational **I**nstitute of **S**tandards and **T**echnology
NPA - **N**on-**P**ower **A**ware
OLB - **O**pportunistic **L**oad **B**alancing
PaaS - **P**latform **A**s **A** **S**ervice
PM - **P**hysical **M**achine
PSO - **P**article **S**warm **O**ptimization

QoS - **Q**uality **o**f **S**ervice
RDPSO - **R**evised **D**iscrete **P**article **S**warm **O**ptimization
SaaS - **S**Software **A**s **A** **S**ervice
SLA - **S**ervice **L**evel **A**greement
SLR - **S**ystematic **L**iterature **R**eview
ThrMmt - Static **Thr**eshold and **M**inimum **M**igration **T**ime
UMC - **U**tilization and **M**inimum **C**orrelation
VM - **V**irtual **M**achine
VMM - **V**irtual **M**achine **M**anager

Chapter 1

Introduction

Contents

This chapter introduces the context and the motivation of the research explored in this thesis, namely resources allocation and workflow scheduling. It starts with the fundamental issues and objectives behind scheduling large-scale workflows in cloud computing environment. The chapter thereafter provides a summary of our contributions.

1.1 Context and motivations

Nowadays, we live in a data-driven world where data-intensive applications are bringing fundamental improvements to our lives in many different areas such as business, science, health care and security, to name a few. This has boosted the growth of the data volumes (i.e., deluge of Big Data) [1] . Big data is not merely data, rather it has become a topic, which involves various tools, techniques and frameworks. Big data technologies are important in providing more accurate analysis, which may lead to more concrete decision-making resulting in greater operational efficiencies, cost reductions, and lower risks for business. Organizations that require dynamic information technology infrastructure are moving to cloud due to its scalability and flexible pricing models. Cloud computing and virtualization technology have revolutionized general-purpose computing applications in the past decade. It provides a wide variety of services to its end users such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS).

Cloud computing services are offered with different levels of quality of service to meet the specific needs of different users. Although many cloud services have similar features (e.g., storage services, computing services, network services, etc.), they differ from each other in their non-functional QoS (Quality of Service) such as time, cost, security, energy consumption, etc. These QoS parameters can be provided as services and by needs, and it should meet the cloud clients' requirements defined in the Service Level Agreements (SLAs). SLAs define the negotiated agreements between service providers and consumers and include Quality of Service (QoS) parameters, such as task execution time. For cloud service providers, enforcing such SLAs is imperative. Lack of such agreements can result in users being moved away from a cloud provider and compromising its growth. However, despite the success of cloud computing for general-purpose computing, existing cloud computing and virtualization technologies face tremendous challenges in scheduling large scale data applications. These applications are covering important aspects of everyday life: health, education, astronomy, research engineering, etc. and are described by a number of interdependent tasks called workflows. A workflow can be described as a set of tasks which is used to complete some business process. This concept has found best use in scientific and business applications for streamlining and improving the performance of the underlying processes. The growing complexity of big data processing problems has raised the use of scientific workflows for performing complex tasks for specific domain applications. Scientific workflows represent the automation of a scientific process in which tasks are organized based on their control and data dependency. The growth of scientific workflows has also spurred significant research in the areas of generating, planning and executing such workflows in cloud platforms.

Many scientific applications are defined as a set of ordered tasks that are linked by data dependencies. A workflow management system is used to define, manage, and execute these workflow applications on clusters in cloud environments. Main issue in workflow management system is scheduling because it is very difficult to identify the available resource from the central pool of resources at runtime [2]. Scheduling, along with many other issues in cloud computing infrastructures, has been extensively studied in the past several decades. Many complex applications in e-science and e-business can be modeled as workflows. Workflow scheduling is the problem of finding a correct execution sequence for the tasks, i.e., execution that obeys the constraints. In other words, the main challenges are to ensure deadlines (very important in real-time inter-

action), budget (pay-per-use cloud model), energy consumption (power saving for data centers), QoS (to guarantee SLA), and fault-tolerance (rescheduling tasks in case of failures) for complex workflow applications. In this context, workflow scheduling is defined as a strategy to decide which (task selection) and where (resource selection) each application task should be executed, and it determines how the input/output data files are exchanged among tasks [3, 4].

As a result, this thesis addresses the problem of scheduling large-scale workflows in cloud computing environments. It investigates a critical issue in cloud computing systems, which is to design a simple and efficient scheduling algorithms. Increasing size of such systems due to their popularity has yielded unprecedented challenges for the design of such scheduling algorithms. We first develop a detailed taxonomy and a comprehensive survey based on state-of-the art workflow scheduling algorithms. Afterwards, a set of workflow scheduling algorithms is proposed. They are tailored for the multi-tenant, elastic, utility-based, and resource-abundant cloud resource model. Furthermore, they are highly successful in generating schedules that are capable of fulfilling a set of QoS requirements expressed in terms of execution time, load balancing, energy consumption and fault-tolerance.

1.2 Issues and objectives

Cloud computing has been developed to remotely store, analyze and process large amount of data. It has recently been brought into focus in both academic and industrial communities due to the increasing features that cloud computing provides such as on-demand computing, elastic scaling, elimination of up-front capital and operational expenses, and establishing pay-as-you-go business model for information technology services [5, 6]. However, existing cloud computing and virtualization technology face tremendous challenges. One of the most challenging issues in cloud computing is scheduling large scale data applications. Scheduling is a fundamental aspect of cloud computing as it allows application parallelization and improved performance. Scheduling algorithms provide benefit to both the cloud user as well as the cloud provider. On one hand, scheduling algorithms can be designed in such a way that they satisfy the QoS (Quality of Service) constraints imposed by cloud client, and, on the other hand, they can be designed to perform load balancing among virtual machines which results into improvement of resource utilization at cloud provider ends. The overall operating performance of a scheduling

system is strongly related to the efficiency of its scheduling algorithm [7].

In general, the process of a workflow scheduling in a distributed system consists of assigning tasks to resources and orchestrating their execution so that the dependencies between them are preserved. This is an important aspect in the efficient operation of the cloud, as various task parameters must be considered for an appropriate scheduling. Scheduling workflows in a heterogeneous context like the cloud is a combinatorial optimization problem, where it is impossible to find an optimal global solution by using simple algorithms. In addition, this problem becomes more difficult in the case where there are several factors to consider, namely: (1) the different QoS requirements imposed by users; (2) the heterogeneity environment and elasticity of available cloud services; (3) the possibility of combining these services to process workflow applications; (4) the transfer of huge volumes of data (datacenters can be geographically spreaded, which can be real issue for workflows with large input data). Therefore, the workflow scheduling problem is considered as an NP-complete optimization problem [8].

Many algorithms have been proposed to schedule large-scale data applications, and these algorithms somehow differ by scheduling factors and parameters [9, 10, 11, 12, 13, 14]. In addition, most of this algorithms fail to (1) provide a deep analysis of task interdependencies to fully exploit parallelism, (2) increment computer system utilization, (3) adapt the number of resources to run each workflow, and (4) incorporate some basic principles of cloud computing such as the elasticity and heterogeneity of the computing resources.

The algorithms developed in this thesis will consider these key features, and more importantly, two sub problems will be considered. The first one is the resource provisioning which consists of selecting and provisioning the compute resources to run all workflow applications. The second one is task scheduling, in which each task is mapped onto the best-suited resource. This will allow schedulers to decide how many VMs to lease, of what type, when to start them and shut them down, and for how long so that the QoS requirements are met while respecting the precedence constraint between the different tasks of a workflow and taking into account multiple QoS criteria such as load balancing, makespan and energy consumption.

1.3 Contributions

This section outlines the contributions from the overall work of this thesis. We propose three scheduling strategies. The first strategy is based on graph theory, it first balances the load between datacenters and afterwards minimizes the overhead of data exchanges. The second strategy aims at reducing the energy consumption of datacenters and guarantee the quality of service (QoS) imposed by users without violating the SLA, using matching game theory. The last strategy aims at improving the task scheduling process in HEFT (Heterogeneous Earliest Finish Time) algorithm in terms of load balancing across virtual machines as well as minimizing the makespan of a given workflow application under a user-specified financial constraint. Our contributions can be summarized as follows:

- In the first strategy, we propose a graph-based model and algorithm for minimizing big data movement in a cloud environment. It consits of three steps. In the first step, we analyzed the dependencies between tasks and datasets to cluster the datasets into different datacenters based on these dependencies. In the second step, we used HPCC benchmark to quantify each datacenter performance in order to assign a load threshold for each datacenter based on its speed of processing and storage capacity. In the third step, we proposed an algorithm to balance the load among datacenters based on the aforementioned threshold.
- In the second strategy, we propose an algorithm named DT-MG (Many-to-One Matching Game for Tasks Scheduling towards Resources Optimization in Cloud Computing). The algorithm assigns tasks to datacenters in optimal manner as well as reduces resource and energy consumption. We have set an upper and lower utilization thresholds, then, we have tried to keep the total CPU utilization of each datacenter between these thresholds. So, when the datacenter load is above the upper threshold or below the lower threshold, the system is unbalanced and some tasks have to be migrated. This strategy includes four phases: tasks scheduling phase using matching game theory; detection of datacenters state phase (over-utilized or under-utilized); tasks selection for migration phase; and tasks migration phase.
- In the third strategy, we propose an algorithm named E-HEFT (Enhancement Heterogeneous Earliest Finish Time). The main idea of this algorithm is to improve the task scheduling process in HEFT (Heterogeneous Earliest Finish Time) algorithm. Generally, the HEFT algorithm achieves high performance and very good tasks execution

time, but its drawback is that it does not take care of load balancing. Our strategy has four phases. In the first phase, we specify the load threshold of each machine based on both processing speed and storage capacity. In the second phase, we define the datasets dependencies in order to store them into different datacenters based on these dependencies. In the third phase, we group the tasks of a workflow (DAG) by level, then we assign a *Rank* value for each task using the rank function of HEFT algorithm. Finally, we schedule these tasks on the best machine using matching game theory.

- A core constituent of Big Data frameworks is the scheduler, which is responsible for scheduling and monitoring the tasks execution, and rescheduling them in case of failures. As the size of clusters used for parallel and distributed computing increases, failures are increasingly common during execution of large-scale applications. This makes fault-tolerance a critical issue for the efficient execution of any application running on such frameworks. Finally, in this thesis, we intend to reach a better understanding of fault tolerance mechanism of Hadoop MapReduce despite failures. We then evaluate the performance of Hadoop MapReduce subjecte to different failures scenarios. Next, we investigated via simulation the impact of checkpointing, which is a commonly used mechanism for managing fault tolerance in distributed systems, interval selection on the performance of Hadoop under various failure probabilities.

1.4 Thesis organization

Following this introduction, the core chapters of this thesis are organized as shown in Figure 1.1 and are derived from several journal papers, conferences and book chapters published during the PhD candidature. This manuscript is divided into 6 other chapters and is structured as follows:

Chapter 2 : Concepts and state of the art

The chapter provides an overview of the literature that covers basic definitions of cloud computing and a survey on scheduling algorithms for scientific workflows in cloud environment. It is made of two sections: In the first section, we present the concept of cloud computing, its service models, its deployment models and its main actors. In the second section, we introduce the concept of workflow and workflow management systems, and we present workflow scheduling strategies that have been proposed for cloud computing platforms in order to systematically and objectively gather and aggregate re-

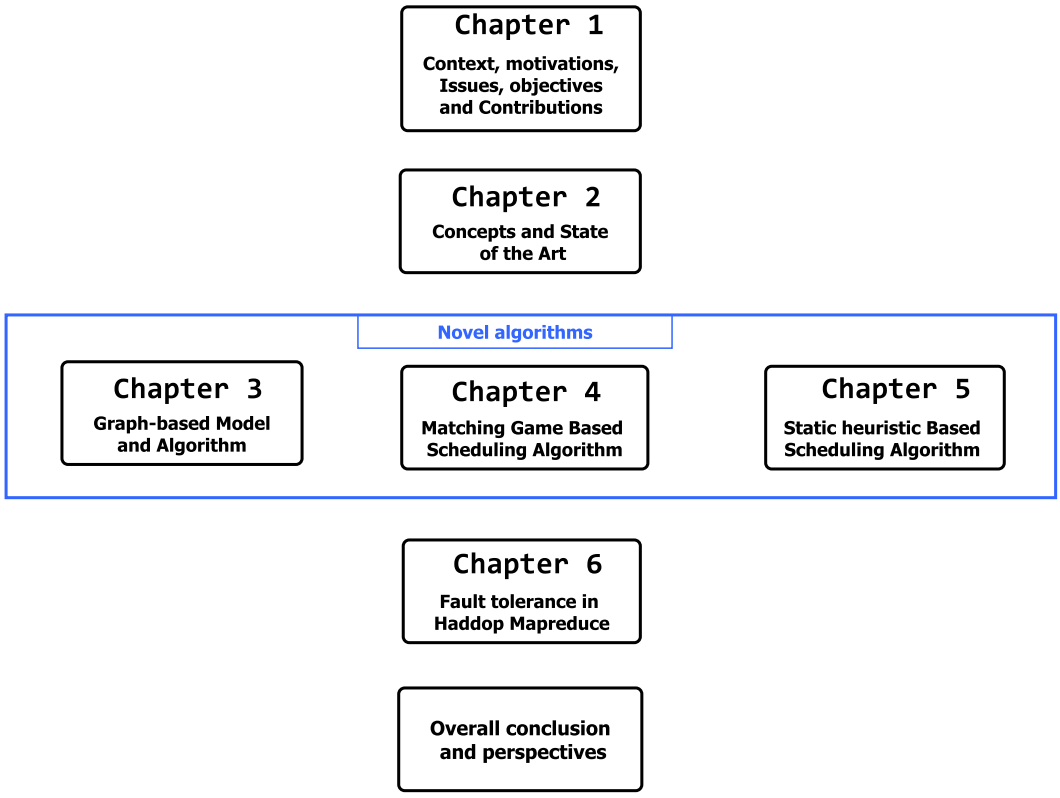


Figure 1.1 – *Thesis organization.*

search evidences about this topic. Then, we present a comparative analysis of the studied strategies. Finally, we highlight workflow scheduling issues for further research.

Chapter 3 : Graph-based Model and Algorithm for Workload Balancing

We describe load balancing and data placement strategies in heterogeneous cloud environments. After that, we deal with the problem of task scheduling in cloud computing. The goal is to optimize multiple QoS metrics, namely load balancing and the overhead of data exchanges. To do this, we present the graph model for big data movement and explain the proposed approach together with some illustrative examples. And finally, we present an experimental evaluation of our proposed algorithm with associated discussions.

Chapter 4 : Matching Game Based Scheduling Algorithm Towards Energy-efficiency

Chapter 4 investigates the resource allocation problem and the trade-off between resource savings and delivered performance in cloud datacenters in order to reduce energy consumption while meeting the QoS requirements. We first present a short background

on the matching game theory covering the different classes of games and their applications, utility function, strategic choice and Nash equilibrium. Then, we present the architectural framework and system model used in this algorithm. After that, we present our strategy for tasks scheduling towards resources optimization in cloud computing. Next, we discuss scheduling performance of our mathematical model and experimental results on Cloudsim simulator.

Chapter 5 : A Static heuristic Based Scheduling Algorithm for Minimizing Makespan

Chapter 5 presents an enhancement of the Heterogeneous Earliest Finish Time heuristic (HEFT) algorithm. Our scheduling algorithm aims at minimizing the total execution time of the workflow, which is viewed as a directed acyclic graph (DAG), as well as to achieve a well-balanced load across all VMs, always obeying both budget and precedence constraints. We start by presenting the state of the art of heuristic algorithms suitable for DAG scheduling. Afterwards, we describe the workflow scheduling problem, its formulation and our objective function. Then, we present our proposed algorithm, assess its applicability and illustrate its performance using Cloudsim simulator.

Chapter 6 : Hadoop Scheduling Performance Under Different Type of Failures

Chapter 6 is dedicated to the failure handling with Hadoop Mapreduce. This chapter begins by presenting a background of Hadoop MapReduce and describing its Fault-Tolerance mechanism. Then, it discusses in detail the types of failure in MapReduce systems and surveys the different mechanisms used in the framework for detecting, handling and recovering from these failures. In addition, it describes the Checkpointing technique as Fault-Tolerance mechanism. After that, it investigates via simulation the impact of checkpointing interval selection on Hadoop performance under various failure probabilities. Then, it analyzes the fault tolerance mechanism of Hadoop Mapreduce under different type of failures by generating a series of failure scenarios for certain type of jobs. Finally, it concludes by a discussion about the open issues and key challenges for providing efficient fault tolerance in MapReduce-based systems.

In the last part, the conclusion, the summary of our contributions, the perspectives brought by these contributions and the appendices are presented.

Chapter 2

Concepts and State-of-the-Art

Contents

This chapter presents in detail the two areas related to the work of this thesis, which are cloud computing and workflow applications. To do this, we conduct a SLR (Systematic literature review) of workflow scheduling strategies that have been proposed for cloud computing platforms. First of all, we present cloud computing and its various components. Next, we investigate a comparative analysis of the studied strategies. Then, we highlight the workflow scheduling issues for further research. The findings of this review provide a roadmap for designing a new workflow scheduling models in cloud computing. This chapter is derived from:

– **Yassir Samadi**, Mostapha Zbakh and Claude Tadonki. "Workflow Scheduling Issues and Techniques in Cloud Computing: A Systematic Literature Review". Springer International Publishing. Cloud Computing and Big Data: Technologies, Applications and Security, 49, pp.241-263, 2018.

– **Yassir Samadi** and Mostapha Zbakh."Big Data Processing on Cloud Computing Using Hadoop Mapreduce and Apache Spark". In K. Munir (Ed.), Cloud Computing Technologies for Green Enterprises (pp. 224-250). Hershey, PA: IGI Global. doi:10.4018/978-1-5225-3038-1.ch009.

2.1 Introduction

Cloud computing is one of the most promising contemporary technologies. It promises to deliver large-scale computational resources over network using a pay as-you-go model [15, 16]. In this model, cloud providers manage resources to process clients' tasks. Furthermore, the requested resources can be scaled up and down dynamically [17].

In addition, cloud computing has emerged as a powerful way to transform the IT industry in order to build and deploy users' applications. These characteristics attract an increasing number of enterprises to run their applications on a cloud platform . Although cloud computing provides all these advantages, it faces many challenges in its development process [18, 19]. According to several surveys, workflow scheduling is one of the main challenges in cloud computing. The term workflow scheduling refers to the mapping of tasks onto resources [20].

Scheduling algorithms yield benefit to both the cloud users and providers. They can be designed to deal with the QoS (Quality of Service) requirements imposed by the cloud clients as well as to perform load balancing among virtual machines in order to improve resources utilization. The overall performance of a scheduling system is related to the efficiency of the scheduling algorithm [7]. The main objective of the scheduling algorithm is to obtain an optimal value such as the lowest execution cost or time.

An optimization problem can be defined as follows:

$$\min f(x) \tag{2.1}$$

Subject to:

$$x \in M = \{x \mid g_k(x) \leq 0,\ k = 1, 2, \cdots, n\} \tag{2.2}$$

$f(x)$ is considered as the objective function, $g_k(x)$ can be regarded as the constraint function. Then, for solving the optimization problem, we can transforme it into minimization problem as stated above.

In this chapter, we investigate different algorithms of workflow scheduling designed for cloud computing environment. Next, each algorithm will be analyzed based on scheduling performance and resources utilization. In our research, we identified 106 papers between 2011 and 2017. Our systematic literature review is organized in four parts:

- **Part 1:** Background on cloud computing: We start by describing some background information concerning cloud computing and its architecture.
- **Part 2:** Workflow scheduling objectives in cloud computing environment: We present the main objectives related to workflow scheduling in cloud. We classify the objectives into 5 main categories as follows: Availability, minimum makespan, maximum resource utilization, security and load balancing.
- **Part 3:** Workflow scheduling techniques in cloud computing environment: We investigate the different types of scheduling algorithm and discuss their impact on the schedulers' performance of such environment. We observe that we can classify the scheduling algorithms used in cloud computing into two main categories: dynamic and static scheduling.
- **Part 4:** Research directions and workflow scheduling issues in cloud computing environment: We present the main issues related to the workflow scheduling in cloud computing. We also present some future research directions for each workflow scheduling objectives discussed previously in part 2 of our review. In addition, we built a road map to enhance the scheduling algorithms discussed in part 3.

2.2 Literature review

In this section, we present the method that we have used to choose the relevant articles. Systematic reviews provide a way to execute in-depth unbiased literature re-

views, aggregating scientific value to the results. The objective of a systematic review is to present a correct assessment regarding a research topic through the application of a reliable methodology. The methodical survey technique described in this chapter has been inspired from [21, 22]. The stages of this literature review include creating a review framework, investigating and recording the review results, and exploring research challenges. Figure 2.1 describes the review technique used in this survey.

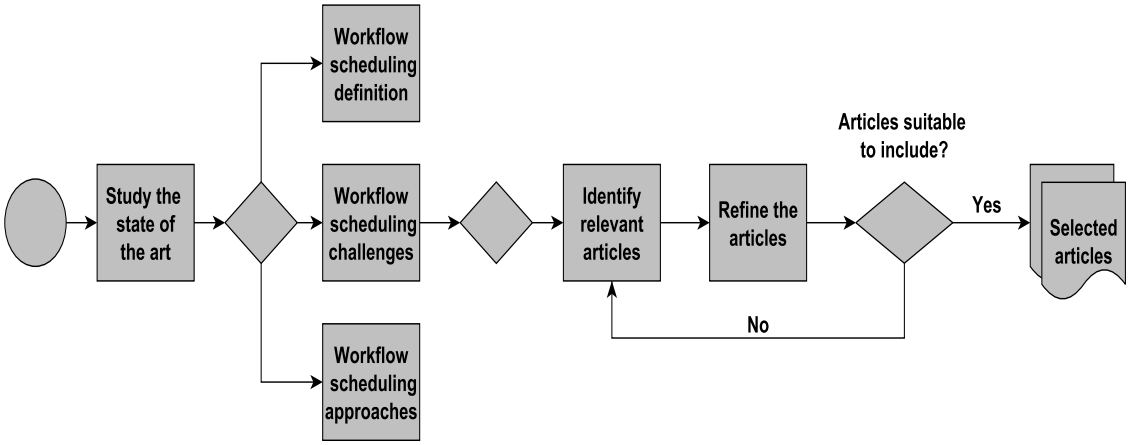


Figure 2.1 – *Review technique used in this systematic review*

2.2.1 Research questions

The present section aims at collecting and investigating all of valuated studies that have examined workflow scheduling challenges and techniques in cloud computing. More specifically, the extraction of salient features and methods will be considered, and their characteristics will be described. The majority of researchers have considered QoS parameters to design the objective functions. We then address the following research questions(RQs):

- What is the current status of workflow scheduling in cloud computing?
- What are the main goals of related researches?
- What kind of validation is performed in each paper? Simulation, analytical model, and experimentation?
- What is the importance of workflow scheduling with increasing use of cloud systems?

- How much are the existing workflow scheduling algorithms meet the main workflow scheduling challenges?
- What QoS parameters are accounted for?
- What simulation tools are used for scheduling and what parameters are they considering?

2.2.2 Data sources

We carried out an electronically based research and considered the following terms: "workflow scheduling", "scheduling challenges", "scheduling techniques", and "cloud computing". Notice that we considered the papers published within the period from 2004 to 2017. We chose this publication period because the field of workflow scheduling in cloud computing has attracted the attention of researchers since 2004. The following digital libraries were used for searching:

- IEEE eXplore (www.ieeexplore.ieee.org)
- ScienceDirect (www.sciencedirect.com)
- Google Scholar (www.scholar.google.com)
- ACM Digital Library (www.acm.org/dl)
- Springer (www.springerlink.com)
- Wiley Interscience (www.Interscience.wiley.com)
- Taylor and Francis Online (www.tandfonline.com)

2.2.3 Quality assessment

Researches in this field started in 2004, but a thorough improvement occurred after 2009. We limited our investigation to various journals, conferences and workshops that have the most elevated quality and considered as the most essential resources in this area. Our search selected 106 potential articles published in peer-reviewed journals and international conferences. After that, we implement a quality evaluation on the most relevant papers based on the criteria of inclusion and exclusion to choose appropriate ones. We manually reduced the number of papers to 70 based on their title, abstract and conclusion. Next, we selected the potential articles focusing only on workflow scheduling in

cloud computing. At last, we used 41 most pertinent articles for this literature review. Figure 2.2 displays the distribution of the selected relevant research papers per year. As can be seen, there is a significant rise in the number of papers on the scope of workflow scheduling in cloud computing environment from 2011 to 2017; also, most of the selected papers were published in 2017.

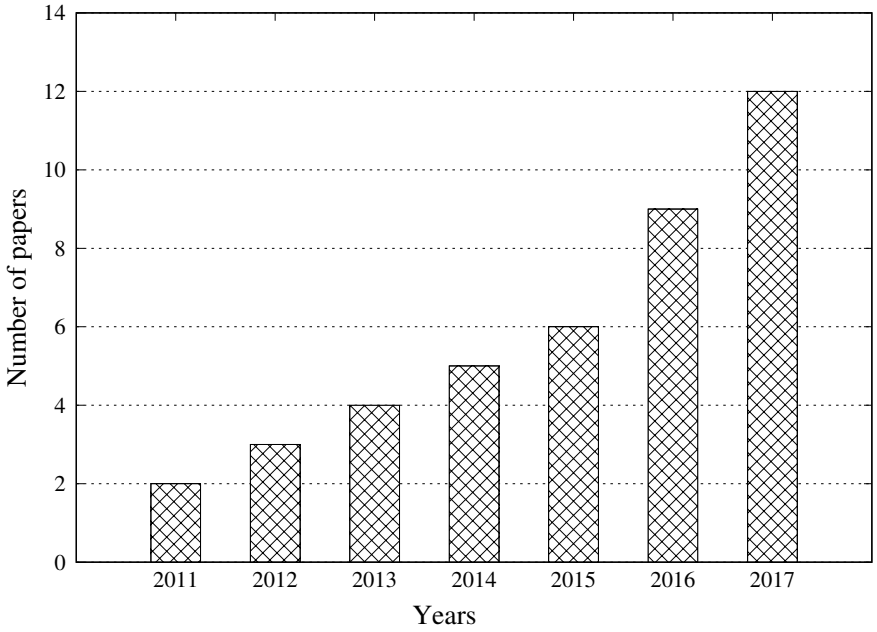


Figure 2.2 – *Number of relevant selected papers per year*

2.3 Background on cloud computing

2.3.1 Definition

Based on the National Institute of Standards and Technology (NIST), *Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction* [23]. The emergence of cloud computing has made a tremendous impact on the Information Technology (IT) industry over the past few years. Cloud providers take care only of the infrastructure management which contains many physical hardwares and softwares [24]. They aim at providing more powerful and reliable cloud platforms, where clients seek to reshape their business models to gain benefits from this new paradigm.

Indeed, cloud computing offers several benefits and features for that makes it attractive to businesses. Some of these benefits include *self-service provision, elasticity, and pay-as-you-go*. Self-service provision allows cloud consumers to access any on-demand computer resource. The elasticity offers the opportunity to increase or decrease the consumption of resources according to the clients' needs. Finally, pay-as-you-go allows enterprises to pay only for the resources consumed. Moreover, cloud computing reduces business risks and maintenance expenses by outsourcing the service infrastructure to the cloud providers [25]. Figure 2.3 presents the main characteristics, the service and deployment models of the cloud computing.

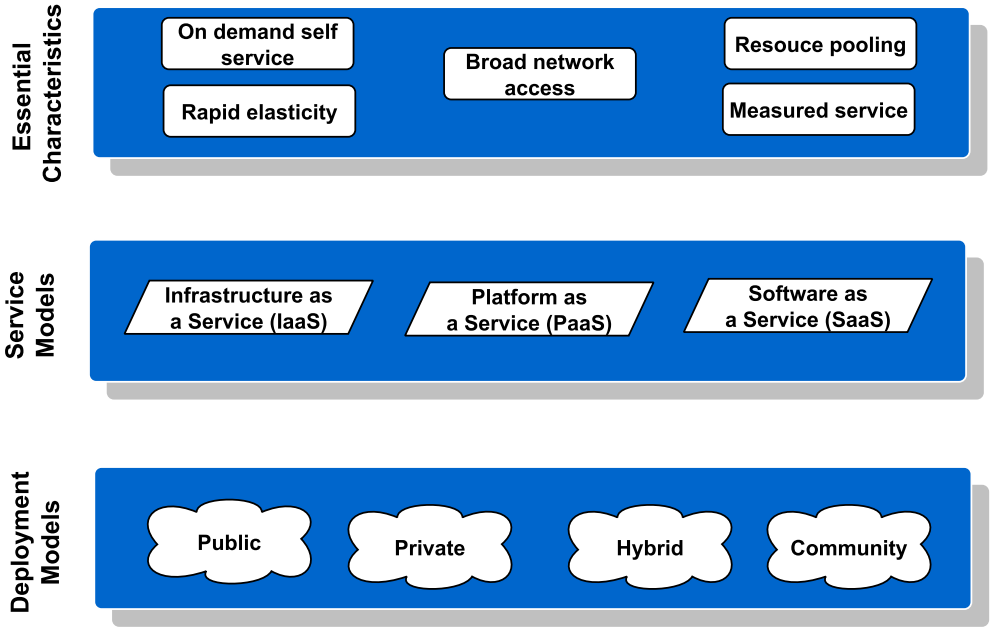


Figure 2.3 – *Visual model of cloud computing definition.*

2.3.2 Main characteristics

Cloud computing provides several features that are different from traditional service computing. Mell and Grance identified and summarized these features [23], which are:

- **On-demand self-service.** The notion of self-service on demand is crucial for cloud users. A user can reserve a set of IT services such as a server and storage network according to his needs. He can also automatically release resources without requiring human interaction.

- **Broad network access.** All resources must be accessible and available to the user simply across the network, regardless of the heterogeneous clients' platforms used (server, PC, mobile client, etc.).
- **Resource pooling.** Resources such as network bandwidth, memory, etc. are pooled to serve multiple users using a multi-tenant model. Most of the cloud providers have a thousand of servers to enable fast load outs. It is often possible to choose an appropriate geographical area to put the data "near" to users.
- **Rapid elasticity.** This feature allows users to quickly provision new resources, so that they can respond to an unexpected rise in load. In other words, the upload of a new instance of a server is done in a few minutes, shutdown and restart in a few seconds. All these operations can be carried out automatically using scripts. It is never easy to foresee the resources that will be needed to set up any computer service, especially when this need is constantly evolving. Cloud computing thus offers a way to provide the computing resources needed for an evolution or peak use of a service.
- **Measured service.** Pay-per-use is a fundamental goal of cloud computing and the desire of companies that use cloud services. The use of resources and services are automatically monitored and controlled, providing transparency for both cloud provider and client. Billing is calculated based on the duration and the quality/quantity of resources used. An off processing unit is not charged.

2.3.3 Service models

Cloud computing can be viewed as a collection of services. NIST (National Institute of Standards and Technology) defines three models of services for cloud computing as depicted in Figure 2.4 [23].

2.3.3.1 Software-as-a-Service (SaaS)

The customer is able to use the provider's applications remotely from the cloud, these applications are accessible from various client devices. SaaS gives clients complete freedom from managing and controlling the underlying cloud infrastructure and the entire software stack. This service enables users to concentrate on using the features of the service to achieve their business objectives. Examples of SaaS providers include Sales-

force.com [1], Rackspace [2] and SAP Business By Design [3].

2.3.3.2 Platform-as-a-Service (PaaS)

The client has the ability to create and deploy on a cloud PaaS infrastructure its own applications using the provider's languages and tools. PaaS provides a ready-deployed software stack that caters to the development and deployment of user's applications on a cloud computing environment. Furthermore, it enables developers to completely focus on application development by eliminating the need for developers to work at the virtual-level. Examples of PaaS providers include Google App Engine [26], Microsoft Windows Azure [4] and Salesforce.com [5].

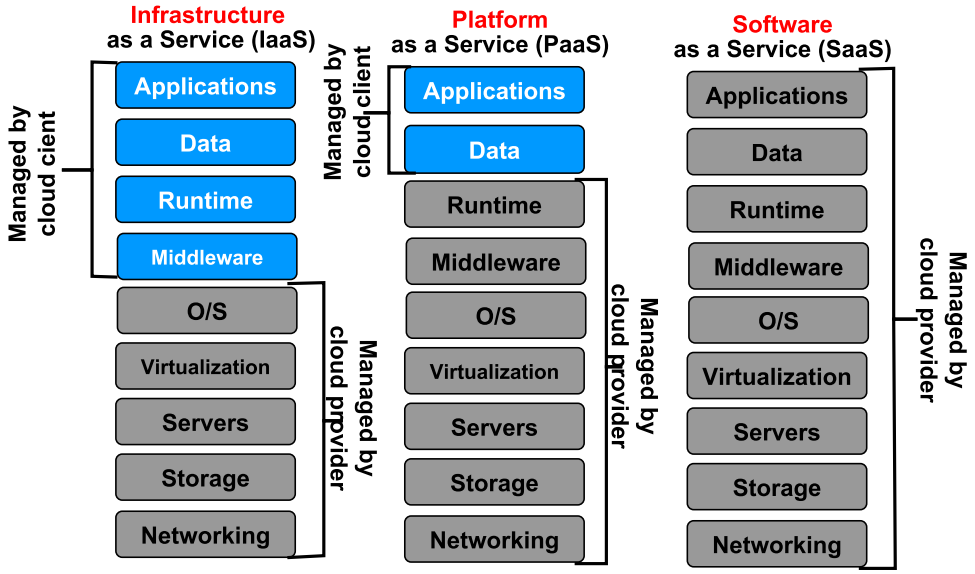


Figure 2.4 – *Cloud Service Models.*

2.3.3.3 Infrastructure-as-a-Service (IaaS)

It provides access to fundamental resources such as physical machines, virtual machines, virtual storage, etc. Cloud user does not manage or control the underlying cloud

[1] http://www.salesforce.com/platform
[2] http://www.rackspace.com
[3] www.sap.com/sme/solutions/businessmanagement/businessbydesign/index.epx
[4] http://www.microsoft.com/azure
[5] http://www.salesforce.com/platform

infrastructure but has control over storage, operating systems, and deployed applications [27]. Cloud user can also choose the main features of network equipment such as load sharing, firewalls, etc. Examples of IaaS providers include Amazon EC2 [6], GoGrid [7] and Flexiscale [8].

2.3.4 Deployment models

Cloud computing types can be classified in terms of who owns and manages the cloud. There are four types of cloud computing type as a common distinction including Public clouds, Private clouds, Hybrid clouds and Community clouds [28]. In Figure 2.5, a general overview of cloud deployment models is represented.

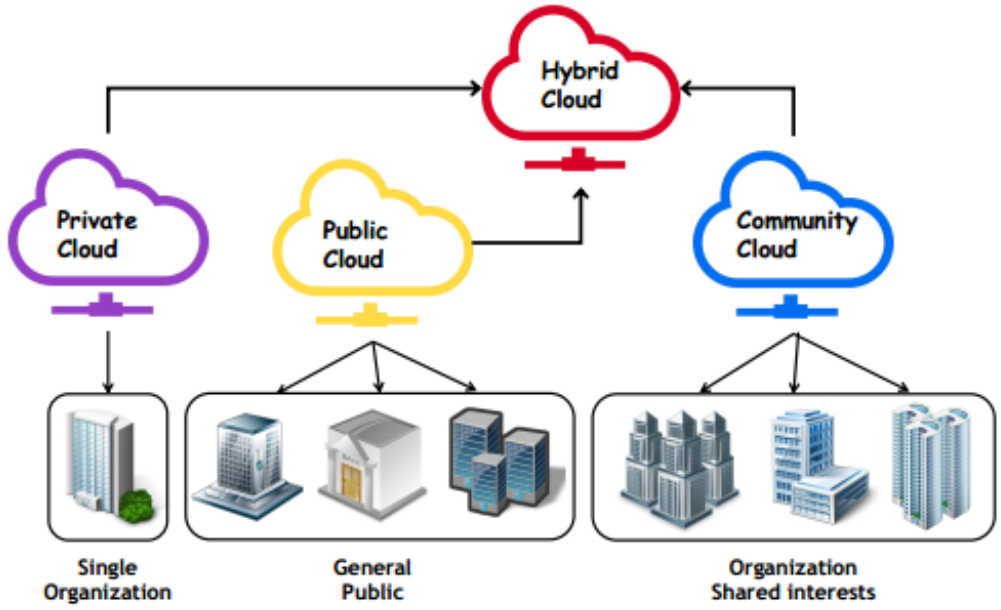


Figure 2.5 – *Cloud deployment models.*

Public cloud:

It allows systems and services to be easily accessible to general public. The cloud infrastructure is rendered available to the general public or a large industrial group and is owned by a company that sells cloud Services.

Private cloud:

[6] http://www.aws.amazon.com/ec2
[7] http://www.gogrid.com
[8] http://www.flexiscale.com

The Private cloud is an infrastructure that allows systems and services to be accessible within an organization. It can be managed internally or by a third party, and hosted internally or externally. It is more secured due to its private nature.

Hybrid cloud:
The Hybrid cloud is a mix of Public cloud and Private cloud. Organizations can perform very important tasks or sensitive applications on the Private cloud, and use the Public Cloud for tasks requiring scalability of resources. Hybrid cloud aims at creating a unified, automated and scalable environment that leverages Public Cloud infrastructure while maintaining full control over data.

Community cloud:
In a Community cloud, the infrastructure of the cloud can be shared by several organizations that have similar requirements, thus increasing their scale while sharing the cost.

2.4 Workflow scheduling in cloud computing

2.4.1 Workflow scheduling problem

2.4.1.1 Workflow scheduling definition

A workflow is an organized collection of tasks related to a global processes. It also defines the order of task to be executed under different conditions, i.e, task synchronization and information flow. In workflow scheduling, applications and services can be decomposed into a sets of smaller components, called subtasks. Different subtasks of a workflow are assigned to resources to meet a pre-defined objectives. There are various applications in bioinformatics, astronomy and business enterprise [29] in which a set of sub tasks is executed in a particular sequence in order to carry out the entire workflow. Generally, a workflow requires a series of tasks to be executed in a particular fashion, which have a parent-child relationship. The parent task should be executed before its child task. A parent task is linked to A child task according to set of rules called constraints [30].
In cloud computing, clients submit their tasks through a client terminal to a scheduler server which works as an intermediate between cloud users and cloud provider. The scheduler is also in charge of initializing tasks and generating task information table including task number, storage space required, task type, etc. Then, the scheduler allocates

tasks to the appropriate resources for task execution according to the scheduling algorithm. Next, when the execution of tasks is finished, the computational nodes return the results to the scheduling server, and the data including computing results and operation information will be sent back to the cloud client. Figure 2.6 illustrates the task scheduling process.

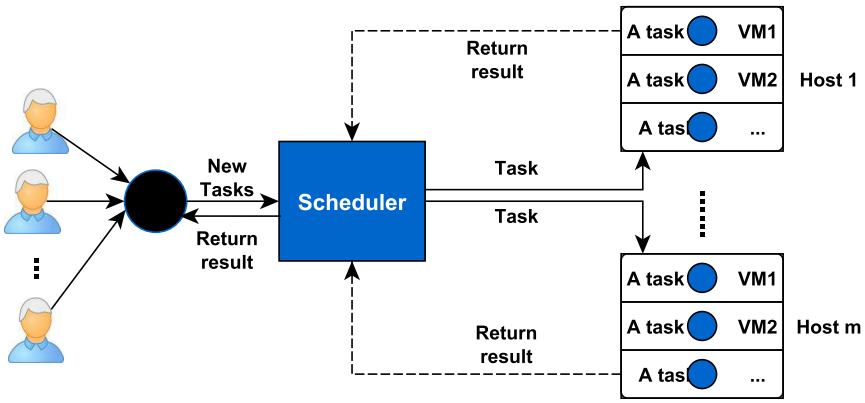


Figure 2.6 – *Workflow scheduling architecture*

2.4.1.2 Workflow scheduling problem

Workflows constitute a common model for describing a wide range of scientific applications in distributed systems [31] [32]. Generally, a workflow is modeled as a directed acyclic graph (DAG) $G = (V, E)$ with n nodes (or tasks), where $t_i \in V$ is a task with an associated computation cost $w_{t_i} \in R^+$, and $e_{i,j} \in E, i \neq j$ is a dependency between t_i and t_j with an associated communication cost $c_{i,j} \in R^+$, case in which t_i is said to be the parent task of t_j and t_j is said to be the child task of t_i. Based on this constraint, a child task can not be executed until all of its parent tasks are completed. If there is data transmission from t_i to t_j, the t_j can start only after all the data from t_i has been received. A task which does not have a parent task is called an *entry* task, denoted t_{entry}, whereas a task which does not have a child task is called an *exit* task, denoted t_{exit}. Generally, there are two types of workflow which are simple and scientific workflows. Figure 2.7 indicates a simple workflow's DAG. It shows a 12-node DAG, where node 1 is the first task to be executed, nodes 2 to 11 can only start their execution after task 1 finishes and sends the data, and node 12 is the last task and can only start its execution after all its parent tasks finish and send their data. Nodes in the DAG are labeled with their computation cost (number of instructions) while edges are labeled with their communication cost (bytes to

transmit).
In addition, there are numerous workflows such as Montage, LIGO, Cyber Shake, SIPHT and Epigenomics applied in astronomy, earthquake researches and so on, which involve complex data of different sizes and need higher processing power. Montage [33] is an astronomy application that was created by the NASA/IPAC Infrared Science Archive as an open source toolkit that can be used to construct large image mosaics of the sky using input images in the Flexible Image Transport System (FITS) format. The CyberShake [34] workflow is a seismology application that calculates Probabilistic Seismic Hazard curves for geographic sites in the Southern California region.

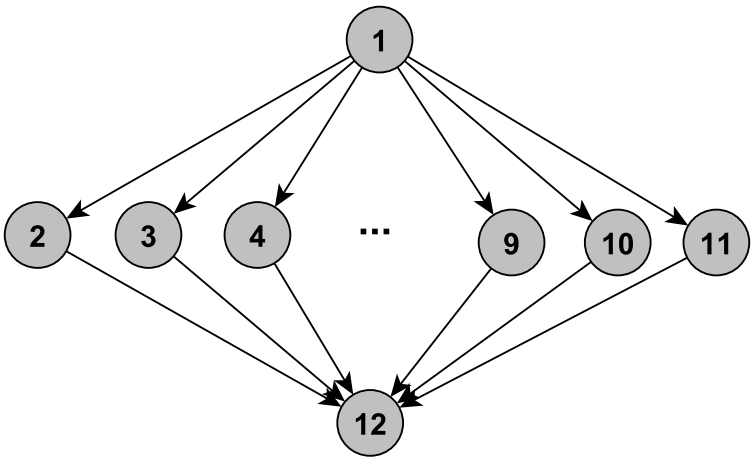


Figure 2.7 – *Example of workflow with 12 nodes*

2.4.2 Workflow scheduling objectives

The main objective of the workflow scheduling is to achieve the expected objectives under constraints by dispatching tasks to the appropriate resource for execution. Currently, the common objectives for workflow scheduling schemes include economic principle, availability, minimum makespan, maximum resource utilization, security and load balancing, etc. [35]. The scheduling objectives discussed in this survey is shown in Figure 2.8.

- **Cost**: The computational nodes may be distributed in multiple locations of the cloud cluster and the cloud client need to pay a management fees to the cloud provider. Total cost incurred by workflow execution in cloud can contain many components such as

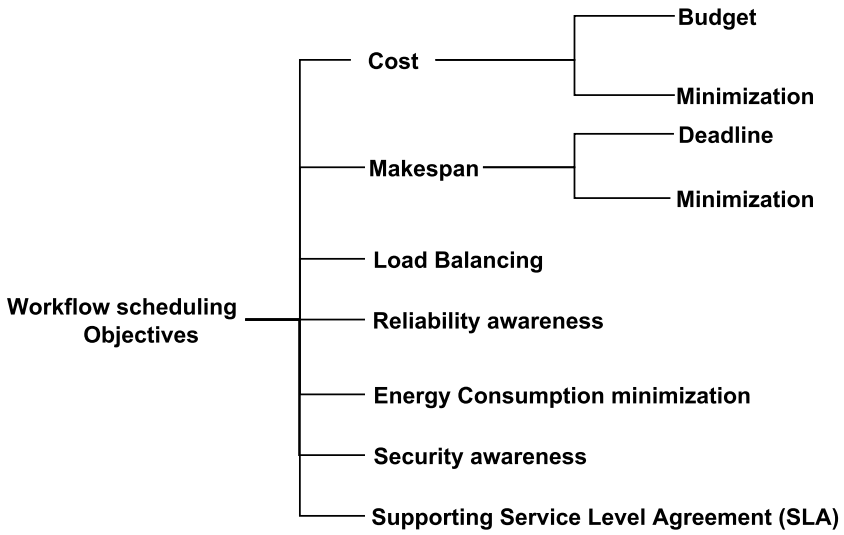


Figure 2.8 – *Workflow scheduling objectives*

compute cost and data transfer cost. Based on [36, 37, 38], the workflow execution cost has become an important objective in cloud workflow scheduling research.

- **Makespan**: is the execution time of workflow, which is mainly determined by the execution time of each task and the communication time between them. In other words, it is the interval between the start time of the first task and the end time of the last task. Makespan of a workflow is dominating objective in most scheduling techniques since the age of cloud computing.
- **Load Balancing**: It is crucial in large-scale data processing applications, especially in a distributed heterogeneous context like the cloud. When a workflow scheduling algorithm aims at scheduling tasks in cloud computing, a scheduler should take the load balancing aspect into consideration to optimize the resource usage and to avoid the overload of any cloud resources.
- **Reliability awareness**: It is the probability that task can be completed successfully within the users' QoS constraints even if resource or task failures occur. For this purpose, some common approaches such as active replications and backup/restart techniques may be applied in the scheduling algorithms. However, they need to be mindful of the additional costs associated with task replication such as waste of time and compute resources [39, 40].
- **Energy Consumption minimization**: The increasing demand of cloud computing motivates the researchers to make cloud environment more efficient for its users and more

profitable for the providers. More and more datacenters are being built to cater customers' needs. However, datacenters consume large amounts of energy and this draws negative attention, which makes energy-efficiency an important concern in cloud computing. Therefore, cloud providers are confronted with great pressures to reduce their energy consumption. A few algorithms have been recently developed which consider a combination of contradicting scheduling goals as they try to find a trade-off between energy consumption performance, and cost. Nevertheless, the authors acknowledge that the energy optimization is still not applicable in virtual machine abstraction level.

- **Security awareness**: Attackers may misuse some cloud features and components to launch specific attacks. So, data security, privacy and governance have become an important issue when an organization decides to deploy cloud computing solution. Thus, high quality security services are increasingly critical for processing workflow applications with sensitive intermediate data. There are many worklfow scheduling approaches proposed to tackle these security issues [37, 41]. They may handle data securely by managing sensitive tasks and data in such a way that either resources or providers with a higher security ranking are used to execute and store them.
- **Supporting Service Level Agreement (SLA)**: The cloud services offered to users consist of a set of components, which may be offered by different providers. SLA is a document that define the negotiated agreements between service providers and consumers which include the Quality of Service (QoS) parameters, such as execution time. Thus, supporting Service Level Agreement is one of the major issues in the current solutions of the cloud. Interaction between cloud user and provider to negotiate SLA is shown in Figure 2.9.

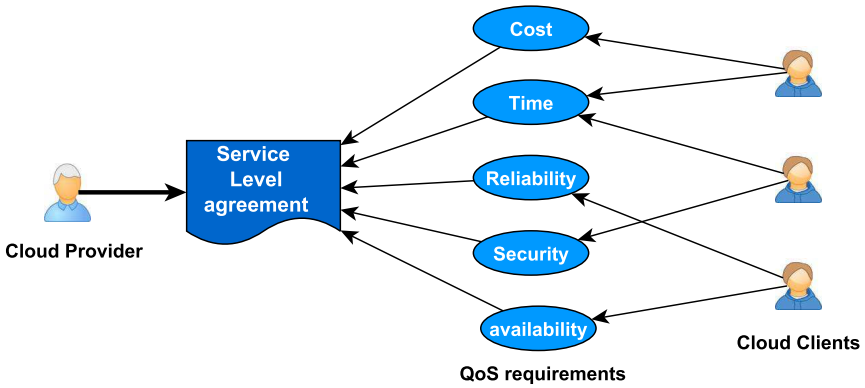


Figure 2.9 – *Process of SLA negotiation*

2.4.3 Workflow scheduling techniques

Scheduling techniques have been developed in response to the evolving cloud technology over the past several years. More information regarding workflows and cloud resources have become necessary in a scheduling process. This section explores several workflows scheduling that used algorithms relevant to each of the objectives presented in the previous section.

2.4.3.1 Cost-aware

Bittencourt et al.[36] presented Hybrid Cloud Optimized Cost scheduling algorithm for workflow scheduling in hybrid environment, where a private cloud is combined with a public one, which aims at optimizing the execution cost. The algorithm schedules workflows in hybrid cloud by first attempting costless local scheduling using Heterogeneous Earliest Finish Time (HEFT) algorithm [42]. If the local scheduling cannot meet the deadline, it decides which resources should be leased from the public cloud and aggregated to the private cloud to provide sufficient processing power for executing a workflow within a given execution time. In the case of selecting resources from the public cloud, authors take into consideration the relation between the number of parallel jobs being scheduled and the cores of each processor.

Authors in [43] proposed a trust service-oriented workflow scheduling algorithm. A trust metric that combines direct trust and recommendation trust is adopted. The weight of cost is incrementally adjusted until the execution time of all tasks satisfies the deadline. It is possible to find an optimum solution with the deadline constraint by adjusting the weights of time and cost effectively. In addition, they provide a balance policies to enable users specifying different requirements, including time, cost, and trust. A case study was conducted to illustrate the value of the proposed technique.

In this paper [9], authors presented a cost optimization algorithm for scientific workflows scheduling on IaaS (Infrastructure as a Service) clouds such as Amazon EC2. Scientific workflows are modeled as Directed Acyclic Graph (DAG). They assume that tasks on each workflow are grouped into levels of identical tasks based on mathematical programming languages (AMPL and CMPL). They formulated the workflow scheduling as a mixed integer nonlinear programming problem to solve the scheduling of large scale scientific applications on hybrid clouds, where the optimization objective is the total cost

with a deadline constraint.

2.4.3.2 Makespan-aware

Authors in [10] have proposed the Intelligent Water Drops (IWD) algorithm which is customized for solving job-shop scheduling problems in cloud computing environment. To increase the diversity of the solution space as well as the quality, five schemes are proposed. In addition, to improve the original IWD algorithm, an improved algorithm named the Enhanced IWD is proposed. The optimization objective is the makespan. Authors demonstrated that the EIWD algorithm can find better solutions for the standard benchmark instances than the existing makespan-based techniques.

Cloud computing raises new challenges to efficiently allocate resources for the workflows and to meet the user's quality of service requirements. To deal with these challenges, Lu et al. [44] proposed an adaptive penalty function for the strict constraints compared with other genetic algorithms. They used co-evolutionary approach to adjust the cross-over and mutation probability. This helps in accelerating the convergence. This algorithm is compared with Random, HEFT, PSO and Genetic algorithms using WorkflowSim simulator on four representative scientific workflows. Experiment results show that the proposed algorithm produced results better than PSO, GA, HEFT and Random scheduling algorithms in terms of total execution time and cost.

2.4.3.3 Load-aware

Authors in [45] examined the reasons that cause runtime imbalance and dependency imbalance in task clustering. Then, a balancing methods are proposed to address the load balancing problem when performing task clustering for five widely used scientific workflows. Task clustering is a runtime process, combining multiple short execution time tasks into a single job, using this process, the scheduling overhead is minimized as well as the runtime performance is improved. Finally, they analyzed the relationship between these metric values and the performance of proposed task balancing methods. Simulation results show that the proposed method gives considerable progress over baseline techniques in terms of load balancing among the set of tasks.

In [11], authors proposed a load balancing scheduling technique for workflows in a cloud environment. The proposed algorithm works in two phases. It first calculates the

priorities of all tasks. Then, it selects virtual machines and schedules tasks. The overall load generated after the execution of current task is also taken into consideration. The simulated results are compared with the benchmark scheduling heuristic named heterogeneous earliest finish time (HEFT) and a variation of the proposed technique. The results show that the proposed approach remarkably display the performance metrics i.e., minimization in makespan and maximization in average cloud utilization.

A balanced scheduler with data reuse and replication for scientific workflows in cloud computing systems was proposed by Israel Casas et al [46]. They have considered that the increment of quantity of resources does not guarantee the reduction in execution time. Next, they proposed a BaRRS algorithm that splits scientific workflows into multiple sub-workflows to balance system utilization via parallelization. BaRRS analyzes the key application features (e.g., task execution time, dependency patterns and file sizes) of scientific workflows for adapting the existing data reuse and replication techniques to cloud systems. They concluded that the optimal number of virtual machines depends on the workflow characteristics. Experimental results prove the superior performance of the proposed algorithm compared to the state-of-the-art scheduling techniques.

2.4.3.4 Reliability-aware

Reliability in cloud computing is how a cloud system is able to consistently provide its services without interruption and failure. However, failures are unavoidable in such large distributed systems. It is also well studied that cloud resources experience fluctuations in the delivered performance. These challenges make fault tolerance an important criterion on the scheduling process. Authors in [47] proposed an adaptive and just-in-time algorithm for scientific workflows scheduling. They used resubmission strategy to find another suitable process unit to re-execute failed tasks. This algorithm uses both spot and on-demand instances to reduce cost and provide fault tolerance.

To model the failure characteristics of a cloud environment, authors in [48] developed a Monte Carlo Failure Estimation (MCFE) algorithm that considers Weibull distributed failures in cloud. Monte Carlo method can correctly model a complex system and give results that are near to complex system operations. This approach can also minimize execution time by using divide and merge pattern for parallelization. Authors proposed a Failure-Aware Resource Scheduling (FARS) algorithm that considers the reliability of task execution while assigning tasks to virtual machines. FARS Algorithm is an extension

of the famous HEFT algorithm. The proposed algorithm is compared with HEFT using cloudsim toolkit using makespan as a performance metric. Results show that FARS algorithm performed better than HEFT.

Reliability is widely identified as an increasingly relevant issue in heterogeneous service-oriented systems because processor failure affects the quality of service. Replication-based fault-tolerance is a common approach to satisfy application's reliability requirement. In [49], authors dealt with this issue and proposed a heuristic replication for redundancy minimization (HRRM) method. It exhibited a significant improvement in resource cost reduction and satisfaction of application's reliability requirement with low time complexity. Experimental results on real parallel applications proved that HRRM algorithm can generate the minimum redundancy with a short execution time in comparaison with the state-of-the-art algorithms.

2.4.3.5 Energy-aware

Energy consumption is one of the major challenges in cloud resource allocation process. Authors in [50] present a virtual machine (VM) placement scheme which tries to minimize the energy consumption based on Particle Swarm Optimization (PSO) algorithm. The main advantage of PSO is combining the local and global search methods to balance the exploration and exploitation. They improve PSO by redefining its parameters, adopting an energy-aware local fitness first strategy to update the particle position and applying a two-dimensional particle encoding scheme. Finally, this improved PSO is used to provide optimal VM replacement by considering the energy consumption metric.

In paper [51], authors proposed a new scheduling approach named PreAntPolicy that consists of a prediction model based on fractals and a scheduler on the basis of an improved ant colony algorithm. This efficient prediction model is developed to assist the algorithm that decides to turn on/off hosts. It helps to avoid performance and energy losses, which is triggered by instantaneous peak loads on account of scheduling. In addition, the scheduler is responsible for resource scheduling while minimizing energy consumption under the Quality-of-Service (QoS) constraints. Experimental results demonstrate that the proposed approach exhibits excellent energy efficiency and resource utilization.

Traditional research in workflow scheduling mainly focuses on the optimization constrained by time or cost without paying attention to energy consumption. Through this

way, Yassa et al. [12] formalized this problem as a multi-objective optimization problem. They solved it using both genetic algorithm (GA) and particle swarm optimization (PSO) algorithm. This approach allows processors to operate in different voltage supply levels by adapting clock frequencies. A compromise between the quality of schedules and energy is involved by this multiple voltage. Simulation results highlighted the robust performance of the proposed technique.

2.4.3.6 Security-aware

High quality of security service is increasingly critical for cloud workflow applications. However, existing scheduling strategies for cloud systems disregard security requirements. To address this issue, authors in [37] introduced a security-aware and budget-aware (SABA) scheduling strategy to minimize the makespan with budget constraint. This strategy holds an economical distribution of tasks among the available CSPs (Cloud Service Providers) in the market to provide customers with shorter makespan as well as security services. Experimental results showed that the proposed scheduling strategy is highly effective under a wide spectrum of workflow applications. Then, Li et al. [52] proposed a security and cost aware scheduling (SCAS) algorithm for workflow application to optimize the execution cost with deadline and risk probability constraints in cloud environment. The proposed approach used the meta-heuristic optimization technique, particle swarm optimization (PSO), the coding strategy is devised to minimize the total workflow execution cost while meeting the deadline and risk rate constraints. Results demonstrated the effectiveness of the proposed algorithm compared to state-of-the-art algorithms.

Adding security services to applications automatically causes overhead in terms of execution time. The trade-off between achieving high computing performance and providing the desired level of security protection imposes a big challenge for workflow scheduling in cloud computing environment. To solve this problem, Arunarani et al [13] proposed a security and cost aware scheduling algorithm for heterogeneous tasks in scientific workflow executed in a cloud. The algorithm is based on the hybrid optimization approach, which combines Firefly and Bat algorithms. Experimental results demonstrated that the proposed algorithm always outperforms the traditional algorithms in terms of minimizing the total execution cost while meeting the deadline and risk rate constraints.

2.4.3.7 SLA-aware

To satisfy the requests of cloud users, services must be provided in accordance with the required level of quality of service (QoS). QoS is the capability to guarantee a definite level of performance based on the parameters described by the cloud client in the Service Level Agreement (SLA). SLA is an authorized agreement that describes QoS in written form. One of the major challenges in the current cloud platforms is to provide the required services according to the QoS level expected by the customer. In this paper [53], authors proposed a SLA-aware PaaS cloud platform called Cloudcompaas. The platform aimed at providing a high-level metrics which are closer to end-user perception, and a flexible composition of the requirements of multiple actors in the computational scene. Moreover, the proposed platform manages the complete resource lifecycle, being able to sustain heterogeneous utilization patterns. This platform could be dynamically adapted to correct the QoS violations by using the elasticity feature of the cloud. The simulation result showed that this solution can achieve minimum cost and maximum efficiency, under highly heterogeneous utilization patterns, for several tested workload profiles.

To tackle the resource allocation problem within a datacenter that runs different types of application workloads, particularly non-interactive and transactional applications. Garg et al. [54] proposed a scheduling technique which considers SLA-based VM management with mix workload. The proposed method predicts the CPU utilization of transactional applications using the Artificial Neural Network (ANN) model. During the under-load of transactional applications, the CPU cycles are stolen and allocated to the batch jobs. This technique does not only maximize the resource utilization and profit, but also ensures that the QoS requirements are met as specified in SLA.

In 2013, Wang et al. [14] proposed an adaptive scheduling algorithm for a hybrid cloud under the desired QoS constraints. They exploited runtime estimation and several fast scheduling strategies for near-optimal resource allocation, which results in high resource usage rate and low computation time in the private cloud. They argued that to maintain QoS in a hybrid cloud, private cloud resources should be maximally utilized which will also reduce usage cost in public cloud. The results showed that the performance of their algorithm is superior in terms of task waiting time, task execution time and task finish time compared with FIFO and FAIR scheduler using CloudSim simulator.

Summarization of these techniques is presented in Table 2.1.

Name of algorithm	Nature of algorithm	Optimization model	Optimization criteria	Tool used
HCOC [36]	Single objective	Heuristic	Cost	Real cloud
TWFS [43]	Multi-objective	Service-oriented	cost and deadline	Simulation
Malawski et al [9]	Bi-criteria	Hybrid HO	Deadline and cost	Real cloud
EIWD [10]	Single objective	Meta-heuristic	Makespan	Simulation
DCGA [44]	Bi-criteria	Genitic algorithm	Deadline and Cost	WorkflowSim simulator
Chen et al [45]	Single objective	Heuristic and task clustering	Load balancing	WorkflowSim simulator
Madhu et al [11]	Bi-criteria	Heuristic	Load balancing and makespan	Real cloud
BaRRS [46]	Multi-objective	Exponential graph based	Execution time and monetary cost	Real cloud
Poola et al [47]	Multi-objective	Heuristic	Makespan, cost and fault tolerance	CloudSim simulator
FARS [48]	Bi-criteria	Monte Carlo method	Reliability and makespan	CloudSim simulator
HRRM [49]	Bi-criteria	Heuristic	Cost and reliability	Real cloud
PreAntPolicy [51]	Single objective	Prediction model	Energy	CloudSim simulator
DVFS-MODPSO [12]	Multi-objective	Meta-heuristic and PSO based	Energy, cost and makespan	Real cloud
SABA [37]	Bi-criteria	Heuristic	Makespan and security	Real cloud
SCAS [52]	Bi-criteria	Meta-heuristic 2 and PSO based	Security and cost	Real cloud
FFBAT [13]	Bi-criteri	Hybrid optimization	Security and cost	CloudSim simulator
Cloudcompaas [53]	Bi-criteria	Dynamic scheduling	Cost and efficiency	Real cloud
Garg et al [54]	Multi-objective	Static scheduling	Resource utilization and QoS	CloudSim simulator
Wang et al [14]	Multi-objective	Heuristic	waiting time and makespan	CloudSim simulator

Table 2.1 – *Summarization of existing workflow scheduling algorithms.*

2.4.4 Workflow scheduling issues

Large-scale scientific workflow applications come up with increasing demands of resources. As the resource demands by large-scale applications deployed on the cloud increase, workflow scheduling in a cloud environment is becoming a challenging task. In addition, most of existing scheduling techniques on the cloud are conducted for relatively simple scenarios. For instance, optimizing cost and time from user's point of view with satisfying the users' QoS requirements is considered the main goal of these techniques. Furthermore, resources provided are fixed and unchangeable during the process of scheduling. There are also some challenges to make workflow scheduling on the cloud more consistent with the real world scenarios.

1. **Dynamic cloud computing environment:** The majority of workflow scheduling algorithms are conducted in static cloud environment with fixed parameters such as the number of virtual machine and network bandwidth. However, cloud computing is a dynamic environment with variation of CPU frequency during the workflow execution. In addition, it is considered as an embodiment of the scalability which enables users to get precise amount of needed resource to execute tasks in an economical way. Therefore, an adaptive scheduling solution should be more effective in solving a real-world problems.

2. **Integrated architecture:** The second challenge is to integrate the workflow management system with cloud cluster. We require a workflow management system that acquires computing resources, managing resources, dispatch tasks, monitors process, and tracking of resource for effective utilization. A workflow engine should be designed with strong functionality to deal with large-scale tasks. Also, it must be user-friendly in which the user can define required parameters easily for workflow scheduling.

3. **Reliability:** Despite the attractive features of cloud platform (in terms of scalability, dynamicity, and low cost), the inherent unreliability of this system has caused great threat to the applications due to failures that may occur during the task execution. Many existing studies on workflow scheduling on the cloud aims at either optimizing the deadline or the cost, ignoring the necessity for reliability. Therefore, in large-scale distributed systems, the scheduling of an application must also account for reliability. Low reliability of cloud will increase scheduling failure rate, thus the makespan and cost will both increase. Two important issues need to be

considered in order to enable reliable workflow scheduling: (i) how to evaluate the reliability of a resource and (ii) how to perform reliable scheduling based on the reliability information of resources. Thus, considering the reliability of the cloud workflow scheduling is a challenge in the future research.

4. **Security:** In QoS challenges, researchers have paid less attention on security aspect than others such as makespan and cost. Privacy protection and data security are vital problems to be solved in scheduling cloud workflows. Therefore, it is of paramount importance to focus on security challenge, which consequently improve the degree of trustworthiness of candidate resources. In addition, security can further improve the robustness and flexibility of workflow scheduling approaches to design an effective solution. Thus, how to protect private data in a cloud may be an issue worth studying.

5. **Big data management and workflow scheduling:** Currently, massive amount of data are carried by cloud platforms. In addition, workflow applications are more data intensive. So, the data resource management, data follow and data transfer between storage and computing resources are the main bottleneck. It is very crucial to find an efficient way to manage data needed by the workflows.

6. **Support for interactive workflows:** An interactive workflow [55] is used to control user navigation, perform view play, and interact with the user through clicking on buttons and hyperlinks. Unlike scientific workflows [56], which can be applied with a complex static scheduling to minimize the makespan [57], interactive workflows are involved with human interactions and mainly consider factors such as response time, stability and security, etc. Since most existing techniques focus on scientific workflows, so it is necessary to focus on other types of workflows which are widely existed in practical applications.

2.5 Conclusion

In recent years, workflow scheduling has evolved to become a critical factor that can significantly affect the performance of cloud computing platforms. This crucial issue is addressed by many researchers. In this chapter, we have presented the cloud computing, a relatively recent paradigm which builds on an economic model based on the actual consumption of users. It has been quickly adopted by many companies to run their ap-

plications. We have explored various research challenges in the cloud. One of these challenges is the scheduling of workflows, which is an essential part of workflow management systems. In addition, we have performed a systematic literature review of the existing algorithms related to this topic. A description and discussion of these algorithms is also included and it aims at providing further details and understanding of prominent techniques as well as further insight into the fields' future directions. Through extensive literature survey, it has been found that there are many algorithms for workflow scheduling, and these algorithms somehow differ in scheduling factors and parameters.

However, the existing algorithms fail to either meet the users' Quality of Service (QoS) requirements such as minimizing the execution time, satisfying the budget constraint of an application, or to incorporate some basic principles of cloud computing such as the elasticity and heterogeneity of the computing resources. To deal with these problems, in the next chapters, we present our contributions on workflow scheduling in cloud computing based on the following objectives: workload balancing, energy consumption and makespan.

Chapter 3

Graph-based Scheduling Algorithm for Workload Balancing

Contents

This chapter proposes a threshold-based load balancing algorithm, which first balances the load between datacenters, and afterwards minimizes the overhead of data exchanges. The proposed approach includes three phases. First, dependencies between the datasets are identified. Second, the load threshold of each datacenter is specified based on both processing speed and storage capacity. Third, the load balancing between datacenters is maintained through the threshold parameter. The heterogeneity of the datacenters together with the dependencies between the datasets are both

taken into account. The algorithm is evaluated with Matlab on various well-known scientific workflows of different sizes. The results show that our algorithm performs better than state-of-the-art algorithms. This chapter is derived from:

– **Yassir Samadi**, Mostapha Zbakh and Claude Tadonki. "Graph-based Model and Algorithm for Minimizing Big Data Movement in a Cloud Environment". International Journal of High Performance Computing and Networking. 2018, Inderscience publisher.

– **Yassir Samadi** and Mostapha Zbakh and Claude Tadonki. "Threshold-based load balancing algorithm for Big Data on a Cloud environment". In Proceedings of the 2nd international Conference on Big Data, Cloud and Applications, 2017, March, p. 18). ACM. Tetuan, Morocco.

3.1 Introduction

The amount of digital data is growing at an exponential rate due to the increase of global sources of data (social networks, Internet of Things, etc.). The quantitative explosion of digital data has forced researchers and developers to find new ways of seeing and analyzing the world. The main concern is to discover new orders of magnitude concerning acquisition, searching, sharing, storing and analyzing data. These huge amount of data should be analyzed and processed efficiently for different purposes in a minimum delay. Several solutions are available to deal with the requirements of big data [1, 58]. Among these solutions, there are distributed data processing tools such as Hadoop MapReduce [59] and Apache Spark [60]. They are mainly used for processing big data, and they work on the principles of parallel computing. So, cloud computing facilitates movement big data, with the main motivation being the need for greater computing capacity and storage.

Cloud computing is a collection of technologies and service models [23] in which the use and the delivery of computing resources such as processing capacity, storage and memory space are all based on the Internet network [61]. This technology can generate considerable economic benefits [62], knowing that on-demand resources are easy to configure and develop on the Internet, where they are easily accessible. Since many years, computer scientists have tried to improve the performance of computers and other IT tools that are used every day. For this purpose, data partitioning and load balancing

have been implemented as important components of cloud computing. On such a heterogeneous platform, task execution time mainly depends on the machine performance. Heterogeneity between two machines can come from the characteristics of the machines, but also come from applications that are executed on these machines. In cloud computing environment, multiple datacenters are usually heterogeneous; each datacenter has its own characteristics and features such as storage capacity and processing speed [63]. Due to the difference between these characteristics, it is inefficient to have all datacenters processing or storing the same amount of data [64].

To achieve good performance and scalability, efficient balancing of the workloads onto distributed and heterogeneous datacenters is necessary. Load balancing in cloud computing improves the performance of the executed tasks on a geographically distributed datacenters through multi-datacenters scheduling of clients' requests. Load balancer [65] is a term commonly used to describe load sharing technologies which receives queries from different clients, or even multiple queries from the same client. Furthermore, in cloud computing environment, a workflow is divided into small tasks, which are assigned to different datacenters in order to achieve a better response time [66]. When a task requires processing data from a geographically distributed datacenters, migration or replication of data becomes a challenge and can yield a significant slowdown on the task execution time, mainly with a huge amount of data [67].

To address the aforementioned problems, this chapter proposes an efficient algorithm to improve the load balance among datacenters while reducing the overhead of data exchanges between these datacenters [68]. Technically, in order to reduce data movement between datacenters, we identify the dependencies between the datasets. If two or many tasks use the same datasets, they should be assigned to the same datacenter. In order to improve the load balancing among datacenters, we assign a load threshold that should not be exceeded in each datacenter based on its processing speed and storage capacity. Therefore, a given datacenter will process more tasks than another datacenter with a lower processing speed.

The remaining content of this chapter is organized as follows: Section 3.3 presents the formulation and our motivation of the studied problem. Section 3.4 explains the proposed approach together with some illustrative examples. Section 3.5 contains the experimental results of our algorithm and associated discussions. Finally, in section 3.6, we draw the conclusion and identify some potential future works.

3.2 Overview of scheduling algorithms

In this section, we discuss some proposed algorithms related to load balancing and data placement strategies. In [69], authors have proposed an algorithm to improve the performance of MapReduce in heterogeneous environments by balancing the load between slow and fast nodes. The proposed algorithm is implemented with Hadoop Distributed File System (HDFS). The datasets were distributed and stored on several heterogeneous nodes according to their computation capacities. The result of the proposed approach gives a better performance compared to other load balancing decisions. However, this approach focuses only on the load balancing in a heterogeneous cluster and it considers only processing speed. Nevertheless, there are another important factors, that we should take into account, which could affect the MapReduce performance such as the storage capacity and the cost of each node in a heterogeneous cluster.

The work in [70] presents a solution to reduce the data movement among geographically distributed datacenters in order to efficiently schedule scientific workflows. In the proposed approach, two strategies are implemented. The first one is an initial data placement strategy to resolve the input data transfer problem based on the dependency and the size of the datasets. The second one is a multilevel task replication strategy for reducing the intermediate data transfers that are generated during runtime stage. To demonstrate the effectiveness of the initial data placement strategy, the results are compared with a random strategy where initial datasets are randomly placed in the available datacenters. To check the effectiveness of the multilevel task replication strategy, the results are compared with "No Task Replication" and "One Level Task Replication" strategies [71]. This approach gives better results via the reduction of data movement up to 15%. However, the load of data processing became unbalanced.

Authors in [72] have proposed a genetic-based algorithm to find the best strategy for moving data among datacenters that can reduce data movement while maintaining load balance. The approach uses a genetic algorithm that minimizes the search time to find an optimal strategy of moving data. Simulation results are compared with the k-means algorithm [73]. The latter, shows that there is no significant difference in the number of data movements between the two approaches. However, in terms of load balancing, there is a large difference between them, in such a way that the proposed approach gives better results than the k-means algorithm.

To find the minimum cost of data aggregation from geographically distributed datacenters on a single datacenter, a graph model approach of geo-distributed datacenters has been proposed [74]. The approach considers a system of geographically distributed datacenters as a directed complete graph. Each node in the graph represents a datacenter. These nodes are connected via high-speed links that represent the edges of the graph. Each link is given a weight, which is the cost of transferring data from one node to another. The result of the simulation shows that the proposed approach gives good results in terms of minimizing the cost of data aggregation among distributed datacenters. However, this approach does not take into account datacenter characteristics except for the cost of storing data. There are other factors that can affect the cost of data aggregation between geographically distributed datacenters such as the storage capacity and the speed of data processing of each datacenter.

The main drawback of these approaches is that they do not deal with data movement and load balancing problems at the same time which can negatively affect the system performance. Authors in [72] have treated both problems. However, the size of the datasets is not taken into account, which does not guarantee that the volume of data movement is reduced accordingly. In some cases, a large dataset may have low dependency but its size will be the dominant factor which can lead to lengthen the data transfer time.

In a cloud computing environment, the data processing capacity of each datacenter can vary from one datacenter to another. Datacenters at high speed must be loaded more than datacenters at low speed. Hence, the efficient use of cloud computing systems requires that the load between nodes should be well-balanced. When the load changes unpredictably during the execution process, a load balancing strategy is required. Furthermore, in scientific workflows, huge volumes of data might be migrated from one datacenter to another geographically distributed. So, a large amount of bandwidth consumption is done because of data migration between datacenters which can deteriorate the system performance. Therefore, if two or more datasets are always used together by many tasks, they must be stored at the same location for reducing the frequency of data movement. Thus, an efficient algorithm to migrate data between geo-distributed datacenters is needed. The major motivation of our work is to simultaneously combine load balancing and the management of large-scale data movement in a cloud computing environment, in order to improve the performance of such systems.

3.3 Problem formulation

3.3.1 Problem statement

Cloud computing platform is generally composed of several datacenters *(DC)* located in different sites. In addition, many instances may be used at the same time to run tasks which may require many datasets. We assume that a cloud cluster is built with a fixed number of interconnected datacenters. Without any loss of generality, we denote the set of datacenters as $DC = \{DC_1, DC_2, \cdots, DC_n\}$, where n is the total number of datacenters in a cloud cluster.

Set $D = \{d_1, d_2, \cdots, d_m\}$, where m is the number of datasets. We also assume that each dataset d_i has two attributes (T_i, S_i), where T_i is the set of tasks that use d_i, and S_i is the size of d_i. A workflow application is composed of several tasks which may require one or multiple input datasets at runtime stage.

3.3.2 Graph model for large-scale data movement

Our approach represents a workflow application as a Directed Acyclic Graph *(DAG)*, $G = (D_s \cup T, E)$, where D_s is the set of existing datasets, T is the set of tasks, and E is the set of edges $\{e_{i,j}, \quad (1 \leqslant i, j \leqslant m)\}$. Each edge $e_{i,j} = (d_i, d_j)$ has a weight, which corresponds to the number of tasks that require dataset d_i and d_j at the same time as shown in Figure 3.1. This weight is equal in both directions for a pair of datasets. Furthermore, the size of datasets might not be the same.

The data movement problem can be defined by first taking the set of datasets and tasks as input, and then finding the dependencies between datasets, and after that, storing them on the set of datacenters based on these dependencies. The data movement strategy aims to reduce the frequency of data migration. The load balancing problem can be defined by quantifying each datacenter in terms of storage capacity and speed of data processing, in order to assign datasets for each datacenter based on its capacity.

3.4 The graph-based proposed approach

In this section, we propose an algorithm to solve the above mentioned problems. The algorithm has three steps. In the first step, we analyze the dependencies between tasks

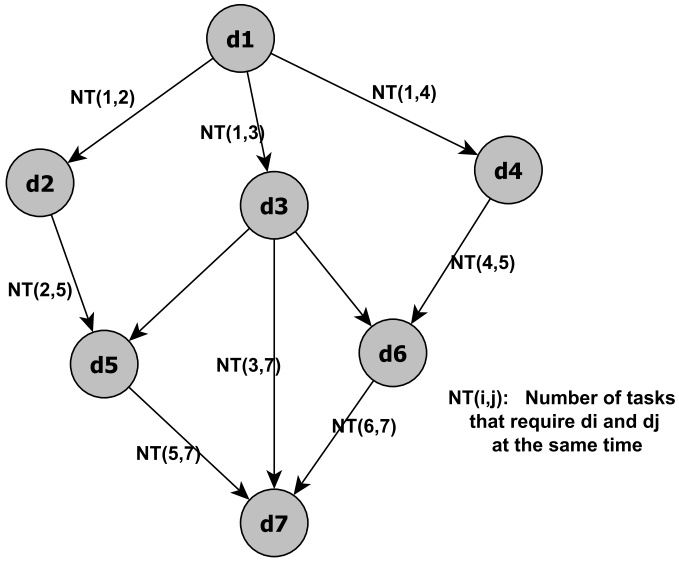


Figure 3.1 – *Graph model of dependency between the set of datasets*

and datasets in order to assign datasets into different datacenters based on these dependencies. In the second step, we specify the load threshold of each datacenter based on its processing speed and storage capacity. In the third step, we balance the load among datacenters based on the load threshold.

3.4.1 Finding datasets dependencies

A data placement strategy is dedicated to efficiently store datasets into datacenters in order to reduce the data movement during the runtime stage. In cloud computing environment, the infrastructure is hidden to users [75], hence, the system decides where to store data. In our strategy, we initially adapt a dependency matrix to represent the affinity between the set of datasets.

Cloud workflows can be complex, the execution of one task might require many datasets. Furthermore, one dataset might also be required by many tasks. We say that two datasets d_i and d_j are dependent on each other if there are tasks that require both d_i and d_j. The dependency degree is the total number of tasks that use both d_i and d_j simultaneously. Mathematically, the dependency degree between the set of datasets is formulated as follows:

$$W_{i,j} = dependency\_degree_{i,j} = card(T_i \cap T_j) \tag{3.1}$$

T_i is the set of tasks that use d_i and T_j is the set of tasks that use d_j. The dependency matrix is defined by: $DM = \{W_{i,j}; \quad (1 \leqslant i, j \leqslant m)\}$. The elements in the diagonal of the matrix DM show the number of tasks that use this dataset. DM is a symmetric matrix of dimension $n \times n$ where n is the total number of existing datasets.

We store datasets that have the highest dependency degree in the same datacenter based on the total number of tasks that use these datasets as shown in formula 3.2.

$$Max(W_{i,j}) \tag{3.2}$$

$$Where \quad W_{i,j} = card(T_i \cap T_j)$$

We proceed as follows :

- First of all, we create a dependency matrix through counting and making in each $(DM_{i,j})$ the value of tasks' number that use d_i and d_j at the same time.
- Then, for placing the dependent datasets with each others, we specify the set of dataset that have the maximum dependency degree.
- We assign the same index for these datasets on L vector. We denote the vector of indices as L in which we put the same value for dependent datasets as shown in algorithm 1.
- After that, we exclude these datasets from the initial set of datasets *(D)* in order not to treat them again.
- Next, in the same way we find the next maximum dependency degree for the rest of datasets and assign to them the same index on the L vector, and so forth.
- We obtain the L vector in which the dependent datasets have the same index as shown in Figure 3.2.
- Finally, we store the datasets that have the same index in L vector in the same datacenter.

Algorithm 1 represents the pseudo code of datasets dependencies algorithm while Table 3.1 displays all the denotations of our algorithms.

As an example, we assume that we have four datasets (d_1, d_2, d_3 and d_4), and we have three datacenters (dc_1, dc_2 and dc_3). We have also three tasks to be processed (t_1, t_2 and t_3), t_1 requires d_1, d_2 and d_3, t_2 requires d_2 and d_3, t_3 requires d_1 and d_4. Based on formula

D	Set of datasets
T	Set of tasks
d_i	Dataset
Mouv	Number of datasets movement among datacenters
L	List of dependent datasets
t_k	Task
DM	Dependency Matrix
NCD	List of non-clustered datasets
Max	Maximum value of dependency matrix DM
NDS	Number of datasets
I	Set of the indices
DCT	List of datacenters threshold
SDC	Data center's storage capacity
d_size	Size of dataset
dd	Set of deppendent datasets
DCU	List of storage capacity of each datacenter that can be used
DCI	List initial of storage capacity of each datacenter that can be used
LDS	List of datasets size
ndc	New datacenter
M	Set of datacenters with initial list of datasets
DCA	List of datacenters' average usage rate

Table 3.1 – *Summary of the main notations used in the chapter.*

3.1, the dependency matrix DM can be presented as follows:

$$\mathbf{DM} = \begin{array}{c} \\ d_1 \\ d_2 \\ d_3 \\ d_4 \end{array} \begin{array}{c} \begin{array}{cccc} d_1 & d_2 & d_3 & d_4 \end{array} \\ \begin{pmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 2 & 0 \\ 1 & 2 & 2 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

We calculate the L vector (Figure 3.2) based on dependency matrix. In other way, we put the same index in the L vector for datasets that have the maximum value in the dependency matrix and so on.

Based on the dependency matrix DM, datasets d_2 and d_3 have the maximum value in the matrix which is 2. So, we put the index 1 in the L vector for these datasets. Then, we move to the next value in the matrix which equals 1 that represents the dependency between datasets $d1$ and $d4$ and we increment the index in L vector by 1. So, we put the index 2 in the L vector for datasets $d1$ and $d4$ as shown in Figure 3.2.

Algorithm 1: Datasets dependencies algorithm

Input: $\boldsymbol{D} = (d_1, d_2, \cdots d_n)$: Set of datasets
$\boldsymbol{T} = (t_1, t_2, \cdots t_m)$: Set of tasks
$\boldsymbol{DM}_{i,j}$: Initalized dependency matrix
Output: L : Vector of depedent datasets
mouv : Nbr of datasets movement.

Step I: creation of dependency matrix ($\boldsymbol{DM}_{i,j}$)
for *each* d_i ***in*** D **do**
  for *each* d_j ***in*** D **do**
    for *each* t_k ***in*** T **do**
      if t_k *needs* (d_i *and* d_j **then**
        DM$_{i,j}$)++

Step II: assigning the same index for the dependent datasets in L vector
$Max = 0, h = 0$,$mouv = 0$ // h: index attributed for each dependent datasets
while $(min(NCD) == 0)$ **do**
  // NCD(i)==0 means that d_i is not clustred yet.
  $h \leftarrow h + 1$
  for *each* d_i ***in*** D **do**
    for *each* d_j ***in*** D **do**
      if $(NCD(i)$ *and* $NCD(j)) == 0$ *and* $Max \leq DM_{i,j}$ **then**
        $Max \leftarrow DM_{i,j}$
      $I = zeros(1, NDS)$
      $I(i) \leftarrow 1 \quad and \quad I(j) \leftarrow 1$
  for *each* d_i ***in*** D **do**
    if $(I(i) == 1)$ **then**
      $L(i) \leftarrow h$ //assigning the same index for the dependent datasets.
      $NCD(i) \leftarrow h$

Step III: Counting datasets movement among datacenters.
for i *from 1 to* NDS **do**
  for i *from 1 to* NDS **do**
    if $((NCD(i) \neq NCD(j))$ *and* $DM_{i,j} \geq 1)$ **then**
      $mouv \leftarrow mouv + 1$

return L and $mouv$.

From the values of the L vector, we conclude that we should store d_2 and d_3 in the same datacenter as well as d_1 and d_4. The number of data movement in this example is equal to 1; therefore, we should move dataset d_1 to the datacenter that contains datasets d_2 and d_3 to execute task t_1.

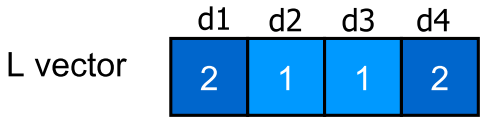


Figure 3.2 – *List of dependent datasets*

3.4.2 Threshold assignment

The threshold represents the storage percentage that each datacenter should not exceed. We will attribute a differential load threshold for each datacenter. In practice, each machine on the cloud does not necessarily have the same characteristics. With differential load threshold's type, each datacenter will take a different load threshold based on its storage capacity and data processing speed. Our allocation relies on the processing speed, but respects the storage capacity constraint.

For quantifying and evaluating the performance of each datacenter, we conside<red the HPC Challenge (HPCC) [1], which is a useful computer benchmarking tool. It consists of seven separate workloads building on the success of the TOP500 list Linpack HPL based workload [2]. The HPC Challenge (HPCC) benchmark suite is designed to give a picture of overall supercomputer performance including floating-point computer power, memory subsystem performance and global network issues [3]. For example, HPC Challenge measures the performance of the system's processors, network bandwidth, memory, and network latency. Authors in [76] showed that HPC Challenge benchmark provides a better understanding of an application performance on the computing systems and it is a better indicator of how high-end computing systems will perform across a wide spectrum of real-world applications.

In our work, we first used the HPL (High-Performance Linpack Benchmark) [77] measurement as a performance value to know the real datacenter performance on the HPCC benchmark. Second, we recorded the execution time of each datacenter based on the output of HPL benchmark. Third, the shortest execution time is used as a reference to normalize the execution time measurements, in such a way that we attributed randomly the highest threshold to the datacenter that has the shortest execution time. Then, the normalized values are used to assign an appropriate load threshold for each datacenter.

[1] http://icl.cs.utk.edu/hpcc/
[2] https://www.top500.org/lists/
[3] http://icl.cs.utk.edu/hpcc/pubs/

As a simple example in Table 3.2, we assume that we have three datacenters on which we run HPC Challenge benchmark. The column "Response time based on HPC Challenge" represents the obtained results. The execution time of datacenters 1, 2 and 3 are 60, 30 and 90, respectively. The execution time of datacenter 2 is the shortest. Therefore, the normalized value of datacenter 2 is set to 1, which becomes a reference used to determine the normalized values of the remaining datacenters. The normalized values of the datacenters 1 and 3 equal the ratio of the response time of these datacenters to the response time of the reference datacenter (datacenter 2). Thus, the normalized values of datacenters 1 and 3 are (60/30=2) and (90/30=3) respectively. Based on column "Normalized Values", we attribute randomly the highest threshold for the datacenter 2, which is 60% in this example. Then, we divide the threshold of the highest speed datacenter (datacenter 2) on the normalized values of datacenters 1 and 3. Thus, the thresholds of datacenters 1 and 3 are (60%/2= 30%) and (60%3=20%) respectively as shown in Table 3.2.

Datacenters	Response time based on HPC Challenge (s)	Normalized Values	Differential Threshold
1	60	2	30%
2	30	1	60%
3	90	3	20%

Table 3.2 – *Allocation of differential thresholds for each datacenter*

3.4.3 Load balancing

Each datacenter is now initialized with a load threshold. For balancing the load among datacenters of the whole cluster, we proceed as follows:

- The load of each datacenter is defined by the sum of the size of the datasets currently stored in this datacenter.
- We make a descending sort for the list of dependent datasets in terms of size.
- We carry out a special descending sort for the list of datacenters in term of storage capacity by prioritizing those that do not contain any dataset.
- For each dependent datasets, we look for a datacenter that has a threshold higher than the size of the dependent datasets.
- When datasets are stored, we check the load of datacenters, or the usage rate of their

resources, to verify that they do not exceed their threshold.

- If the threshold is not exceeded, we continue to store the rest of datasets.
- When the threshold is exceeded, we migrate the smallest dataset among the set of dependent datasets that have saturated the datacenter to another one with a low workload.

Algorithm 2 shows the pseudo code of load balancing among datacenters.

Algorithm 2: Load balancing algorithm

Input: $\boldsymbol{DC} = (dc_1, \cdots, dc_n)$: Set of datacenters
$\boldsymbol{D} = (d_1, d_2, \cdots d_n)$: Set of datasets
DCT : List of datacenters threshold
SDC : datacenter's storage capacity
LDS : List of datasets size
L : List of dependent datasets
Output: ***M***, ***Sdt***: Std of the set of datacenters.

Attribute a load threshold for each datacenter.
Step I: Specify the storage capacity that can be used for each datacenter
for *each dc* ***in*** *DC* **do**
    $DCU(dc) \leftarrow SDC(dc) * DCT(dc)/100$
Descending sort of the dependent of datasets.
Special descending sort of datacenters.
Step II: Storing the set of dependent datasets into the same datacenter
for *each (dd* ***in*** *L vector)* **do**
    while *((dc in DC) and (the set of dd are not stored yet))* **do**
        if $(\sum_{dc \in L} (LDS(dd(dc))) < DCU(dc))$ **then**
            storing the list of depedent datasets into datacenter $M(dc)$.
            $DCU \leftarrow DCU(dc) - (\sum_{d \in L} LDS(dd(d))$
        if $(\sum_{d \in L} LDS(dd(d)) > DCT(DC))$ **then**
            Transfer the smallest dataset from the list of depedent datasets that has saturated the datacenter to another datacenter with a low workload
Step III: Compute the standard deviation of the set of datacenters (*M*)
for *each dc* ***in*** *DC* **do**
    $DCA \leftarrow DCU(dc)/DCI(dc)$
    $Std(DCA)$
return M and Std.

As a simple example shown in Table 3.3, we assume that we have four datacenters

with different characteristics and ten datasets with different sizes. For simplicity, we also assume that the storage capacity of each datacenter is 100 GB. We attribute a load threshold to each datacenter based on step 2 "Threshold assignment" of our proposed approach.

We assume that the result of datasets dependencies algorithm is as follows: each list contains the datasets that have the same index in L vector. So, the first list contains datasets d_1 and d_2 . The second list contains d_3, d_4 and d_5 . The third list contains d_6 and d_7 . In the last list, we have datasets d_8, d_9 and d_{10}.

- $L_1 = \{d_1, d_2\}$
- $L_2 = \{d_3, d_4, d_5\}$
- $L_3 = \{d_6, d_7\}$
- $L_4 = \{d_8, d_9, d_{10}\}$

Before applying our load balancing Algorithm, we store the list of datasets in the set of datacenters. For example we store the first list in DC1, second list in DC2, third List in DC3 and fourth list in DC4 as shown in Figure 3.3.

Datacenters	Datacenters' load thresholds	Datasets	The size of different Datasets (GB)
DC1	60%	Dataset1	10
		Dataset2	14
		Dataset3	20
DC2	64%	Dataset4	10
		Dataset5	30
		Dataset6	14
DC3	70%	Dataset7	38
		Dataset8	18
		Dataset9	21
DC4	50%	Dataset10	11

Table 3.3 – *Characteristics of small Cloud with four datacenters and ten datasets*

It appears that the datacenter DC4 is more over-loaded than other datacenters. In order to overcome this problem, we apply our load balancing algorithm. We first do a descending sort of the list of dependent datasets in terms of size. After that, we carry out a special descending sort of the list of datacenters in terms of storage capacity by prioritizing those that do not contain any dataset. Then, we assign the list of depen-

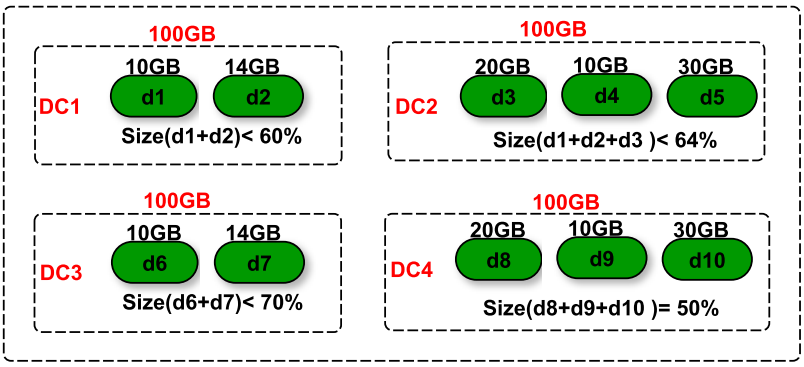


Figure 3.3 – *Initial placement of datasets*

dent datasets with highest size (second list) to the datacenter that has the highest storage capacity (DC3) and so forth. Figure 3.4 shows that the load of datacenters became well-balanced.

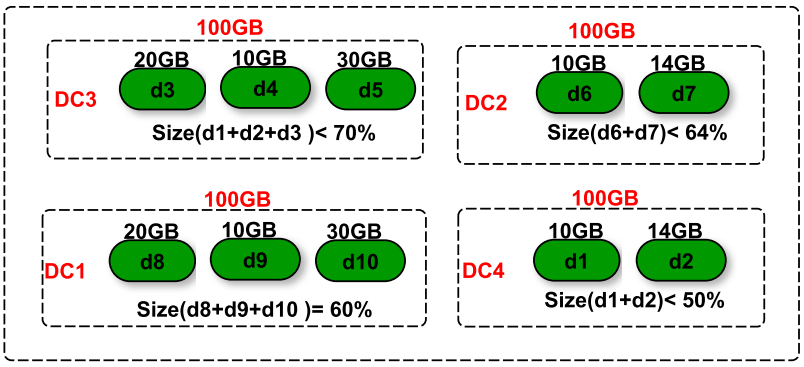


Figure 3.4 – *Clustering the dependent datasets into datacenters based on load balancing algorithm*

According to this example, once the datasets have been distributed within different datacenters, the analysis of resources' usage level of datacenters has been carried out before applying our load balancing algorithm. We notice that the load among datacenters is unbalanced. In fact, the aim of our work is to ensure the set of datacenters are not over-loaded, or even in an idle state while minimizing data movement. As shown in Figure 3.5, datacenters DC2 and DC4 are more loaded in comparison with other datacenters. So, before storing datasets in datacenters, we carry out a descending sort of the list of datasets and a special sort of the list of datacenters as mentioned in subsection 3.4.3. Thus, an accurate load balancing of data distribution among datacenters is obtained on the cluster as shown in Figure 3.6.

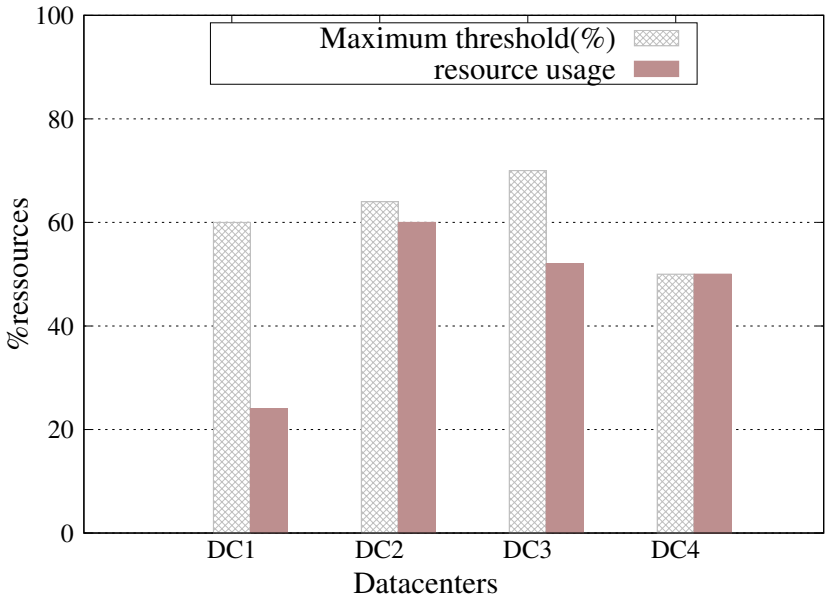


Figure 3.5 – *Initial data placement*

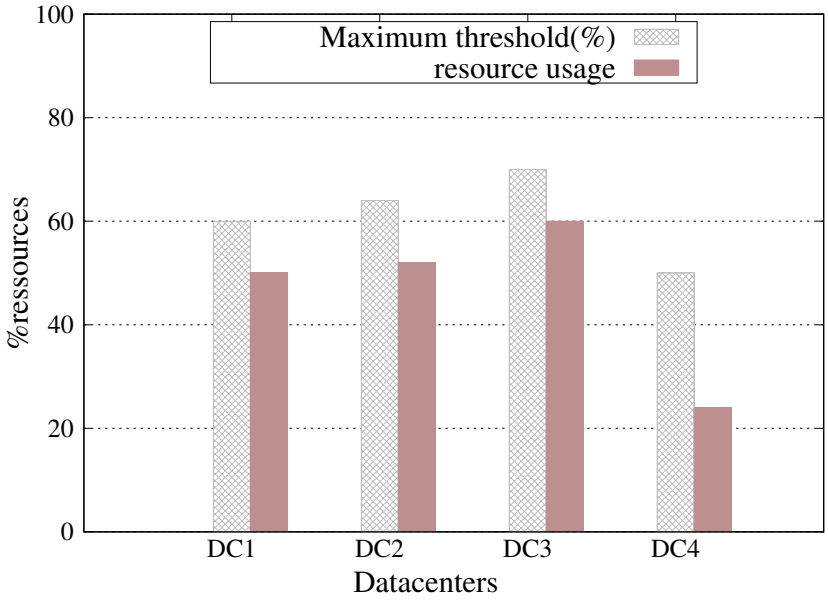


Figure 3.6 – *Load balancing between datacenters*

3.5 Experimental evaluation

3.5.1 Experimental settings

To carry out the evaluation as objective as possible, we first consider a randomly generated scientific workflows. A random workflow graph generator was implemented to

generate DAGs with various characteristics specified by several input parameters, including: (1) number of tasks in the graph; (2) number of input datasets; (3) range of execution time for the tasks; (4) range of data size for input datasets and output datasets; (5) maximum dependency degree between datasets and tasks in the graph. All of the experiments are implemented in Matlab R2009b and tested on an Intel(R) Core(TM) Dual i5-4200M CPU 2.50 GHz with 4G RAM. We carried out the simulation on 20 datacenters as the total number of datacenters that we can use in this experiment. We limited the amount of storage that the datacenters had available during the experiment in range [500 GB, 620 GB]. The sizes of input datasets are set by the range [1GB, 20GB]. Such parameters configuration may not be realistic in the real world but the only aim of this configuration is to demonstrate the quantitative nature of our work. We generate randomly the number of datasets as input of each task on the workflow. Each task can take a maximum of 10 datasets as input. We can control the complexity of the simulation by changing the number of datasets. Every dataset will be used by a random number of tasks. We can also control the complexity of the relationships between datasets and tasks by changing the range of this random number. To prevent biasing to a particular strategy, we ran 10 tests under the same configuration and took the average value as the final result. Then, the simulation software calculated the total data movement amount on the datasets layout.

3.5.2 Performance metrics

Our proposed approach aims at balancing the load between datacenters as well as to reduce the overhead of data exchanges. Therefore, to evaluate the performance of our algorithms we used the following two metrics:

3.5.2.1 Total data movements

The traditional way to evaluate the performance of a workflow system is to record and compare the execution time. However, in our work we count the total data movement instead. Note that to obtain the execution time, we need the processor speed, bandwidth, data management and scheduling strategies as well as other parameters. Since the main objective of the data placement algorithms is reducing the total amount of data transferred, we directly take the number of datasets that are actually moved during the runtime stage as a measurement to evaluate the performance of our algorithms. Next, we compare the experimental results with Genetic [72] and k-means algorithms [73] which

serve as a benchmarking references. In a cloud computing environment, if the total data movement has been reduced, the execution time will be reduced accordingly. Furthermore, the cost of data transfer will decrease too.

3.5.2.2 Index of load balancing

The next set of experiments test the quality of the clustering in terms of accuracy and standard deviation. Standard deviation is related to how much variation from the average (mean) is caused by clustered data. In our case, the standard deviation is the square root of variance of the storage capacity used in each datacenter.

3.5.3 Performance evaluation

Each computation was performed ten times. Thus, the average result values of running 10 different types of workflows with the same parameters are reported in this paper for both data dependency and load balancing algorithms.

We ran the test workflows on 20 datacenters. We used different numbers of datasets in the range of [40, 120] as shown in Figure 3.7a. We can see that when the number of datasets increases, the data movement becomes more complex and its total number increases too. Figure 3.7b shows the data movement among datacenters when we run workflows on different numbers of datacenters.

From the results shown in Figure 3.7, we can observe obvious advantages of our data placement strategy (GM) compared to other strategies GA and k-means [72, 73] in terms of total data movement. When we increase the number of datasets, data movement increases too as shown in Figure 3.7a. However, it can be seen that our approach (GM) is always better than the other strategies. Our approach caused 100.33 movements of datasets on average whereas the GA and k-means algorithms produce 125.22 and 129.55 datasets movements on average, respectively. Our algorithm reduces the amount of data movement by 19.87% and 22.55% in comparison with GA and k-means algorithms, respectively. In the second experiment, we fixed the total number of datasets to 100 and we varied the number of datacenters within range [5, 20] as shown in Figure 3.7b. From this figure, we observe that all data placement strategies continuously increase the total amount of transferred data among datacenters with the increase of the number of datacenters (from 5 to 20 with 5 as the increment). This is because of the increasing number of datacenters which means that the original input datasets are distributed more sparsely, so

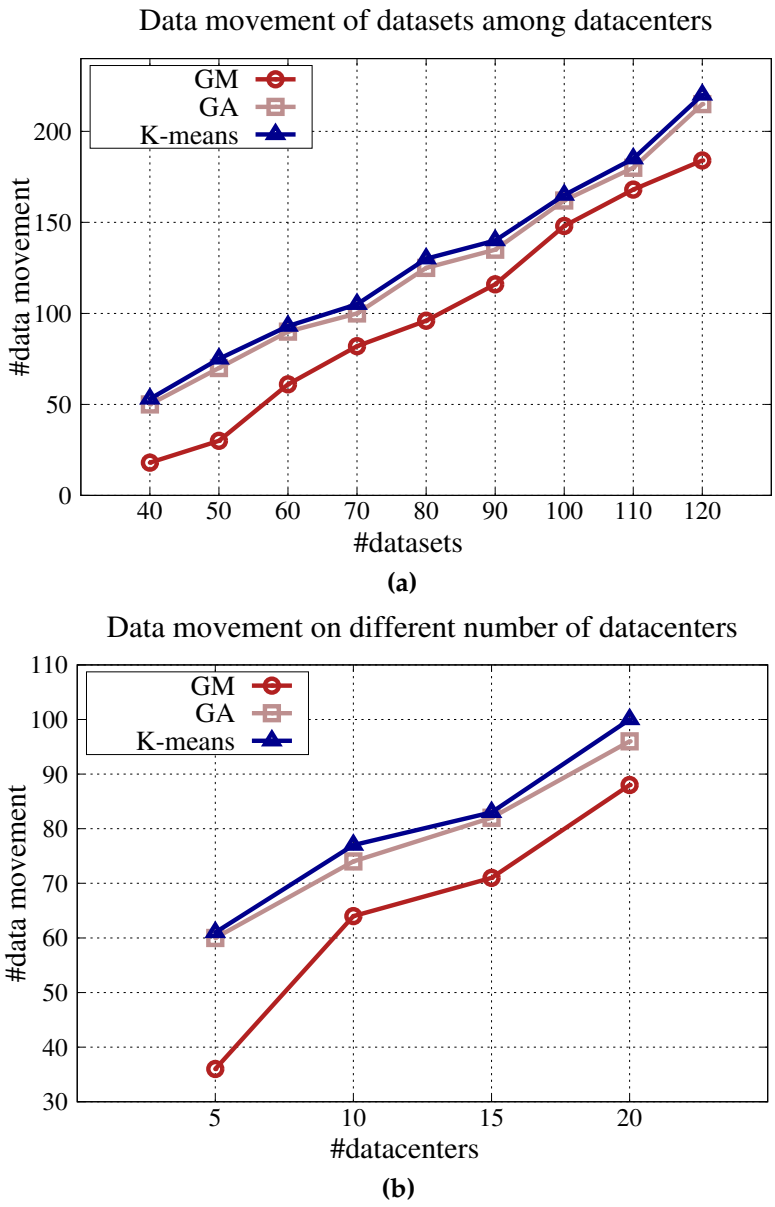


Figure 3.7 – *Data movement among datacenters with different number of datasets and datacenters*

a specific dataset may be placed on a wrong datacenter with a relatively higher probability, thus more data movement is required. The results show that our algorithm is better than GA and k-means algorithms in reducing data movement at different numbers of datacenters. On average, compared with GA and k-means algorithms, 18.8% and 21.12% amount of data movement have declined respectively.

We attribute the efficiency of our proposed strategy to the following reasons: First, our strategy clusters the input datasets according to their dependency and then places

the clustered data on more powerful datacenters with higher priority (subject to storage limitation). Therefore our strategy helps: (1) to aggregate the datasets that are required by the same tasks, so when tasks are scheduled, no required datasets or just a few are necessary to transfer from some datacenters to another ones; (2) to more likely store datasets in datacenters with higher computation capability, which makes powerful datacenter processes more data/tasks, and hence shortens the whole execution time. Second, our strategy takes data size into consideration when calculating the weight of dependencies, therefore it transfers less volume of data compared to other strategies.

Figure 3.8 shows the average standard deviation of running 10 test workflows with different number of datasets on 20 datacenters. We could conclude from the result that our algorithm (GM) can effectively balance the load among datacenters compared to the K-means algorithm and has almost the same result with GA algorithm mainly in the case of large number of datasets. It is well known that the K-means algorithm is a standard clustering algorithm. Datasets are relatively concentrated in few datacenters which leads to let a lot of datacenters to be idle [78]. This is the reason that the index of load balancing of K-means algorithm is very high.

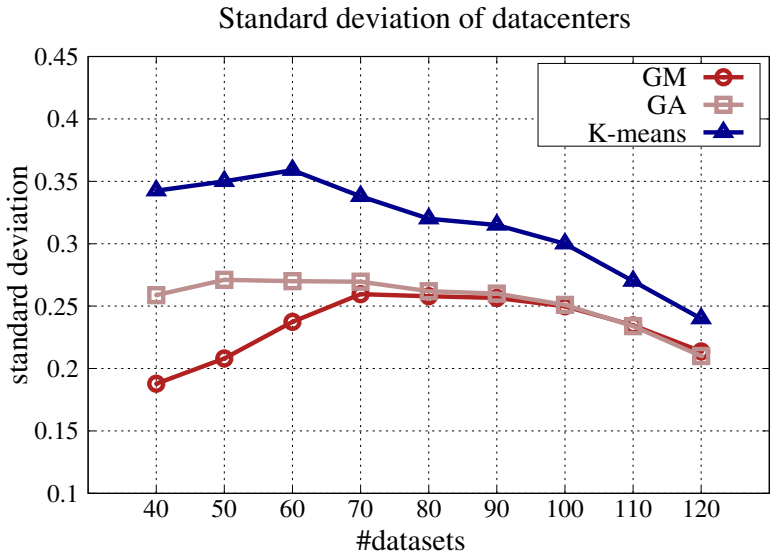


Figure 3.8 – *Standard deviation of workload among datacenters with different number of datasets*

The next experiment is undertaken to measure the CPU time consumption for clustering the datasets into the set of datacenters. The CPU time consumption is timed as "how long it is necessary for the clustering algorithm to converge". Given that the datasets are of fixed volume, CPU time consumption is related directly to the speed of the clustering

operation.

Our algorithm compares data in $O(n^3)$ times in each iteration, so it takes a lot of time to converge in comparison with K-means and Genetic algorithms as shown in Table 3.4. The traditional K-means and genetic algorithms converge easily to a local optimum, so the index of load balancing is worse than our approach as shown in Figure 3.8.

CPU time consumption(s)			
Algorithm	Best	Worst	Average
GM	46.15	53.425	49.18
K-means	34.896	38.963	36.987
GA	27.245	31.621	29.63

Table 3.4 – *CPU time consumption for clustering the datasets into the set of datacenters*

3.5.4 Discussion

In K-means clustering [60], the datasets are placed based on the dependency between them and the target datacenter. But it can happen that many datasets have high dependency on one datacenter. This might result in a noticeable unbalanced load among datacenters. In the data placement strategy based on Genetic Algorithm, the placement strategy is represented as a gene. Initially, they consider that each datacenter is assigned a single dataset. After the mutation step, the correctness of the gene is checked to verify if it is in accordance with the fitness function or not. That means some gene code may be invalid or unacceptable, which can overload some datacenters [79]. When one datacenter is overloaded, K-means clustering and Genetic Algorithms need to reallocate the datasets to other datacentres. The reallocation will cause extra data movement, and will also delay the execution of the workflow.

In our proposed approach, we specify the load threshold of each datacenter based on both the processing speed and the storage capacity. We placed the datasets based on these thresholds, which can effectively balance the load and reduce the number of data movement among datacenters compared to other algorithms.

3.6 Conclusion

In this chapter, we addressed the problems of big data movement and load balancing among datacenters in a cloud environment. We first proposed an algorithm based

on dependency matrix for storing datasets that have a maximum dependencies in the same datacenter. Then, we used HPC Challenge benchmark to quantify each datacenter performance, in order to estimate a load threshold for each datacenter, taking into consideration its processing speed and its storage capacity. To improve the distribution of the load among datacenters, we proposed a threshold-based load balancing algorithm. The results show that our approach gives good results in both load balancing and data movement among datacenters.

The increasing demand of cloud computing motivates researchers to make cloud environments more efficient for its users and more profitable for the providers. More and more datacenters are being built to cater customers' needs. However, datacenters consume large amounts of energy, and this draws negative attention. Therefore, cloud providers are confronted with great pressures to reduce the energy consumed by datacenters. To address this issue, we introduce, in the next chapter, an energy-efficiency approach to deal with the energy consumption problem. This approach reduces the amount of active nodes required to process a heterogeneous workload without degrading the service level in order to reduce the energy consumption by datacenters at the runtime stage.

Chapter 4

Matching Game Based Scheduling for Energy Efficiency

Contents

In this chapter, we propose an energy-aware allocation algorithm named DT-MG, which aims at reducing energy consumption and maximizing the efficiency of the available resources. First, we use the Matching Game Theory model for assigning tasks to datacenters. Then, we study the optimal usage of the resources by migrating all tasks of the physical machines under sub-regime to another ones, followed by their systematic switch to standby mode. Experimental results show that our proposed approach reduces the energy consumption and the number of task migration while maintaining the offered service level in comparison with some existing techniques. This chapter is derived from:

– **Yassir Samadi**, Mostapha Zbakh and Claude Tadonki. "DT-MG: Many-to-One Matching Game for Tasks Scheduling towards Resources Optimization in Cloud Computing". International Journal of Computers and Applications. 2018. Taylor and Francis publisher.

4.1 Introduction

Cloud computing is a newly emerged computing platform that builds on the latest achievements of diverse research areas [23]. On a cloud computing platform, resources are provided as services and by needs, so it should meet the cloud clients' requirements defined in Service Level Agreement (SLA). The cloud clients are interested in reducing the overall execution time of their tasks. From the cloud computing point of view, providing a short response time for parallel jobs being executed on thousands of machines raises a significant scheduling challenge. Many problems about scheduling tasks on heterogenous systems are NP-hard [80], and many meta-heuristic algorithms have been proposed to solve it. The problem of task scheduling in cloud computing can be considered as assigning optimally a set of tasks to the available resources to meet the cloud clients' requirements.

Generally in cloud computing, a file is split into many small blocks which are stored on servers [81]. For fault tolerance, all blocks are replicated and distributed on the cluster. To process the file, the scheduler divides a job into several tasks, each of which is assigned to a server in order to simultaneously process a block of files [82]. In this mode, all tasks can run in parallel to accelerate the execution of the whole job. Furthermore, if a task reads its block from the local server disk, it is called data-local; otherwise, the

task is called data-remote if it retrieves a copy of its block from a remote datacenter. Data transfer in cloud computing has a significant impact on system performance such as increased execution cost and time. In addition, it is difficult to manually assign tasks to heterogeneous computing resources [83, 84]. The increasing demand of cloud computing motivates researchers to make cloud environments more efficient for its users and more profitable for the providers. More and more datacenters are being built to cater customers' needs. However, datacenters consume large amounts of energy, and this is a serious concern. Hence, cloud providers are confronted with great pressures to reduce the energy consumed by datacenters. To address this issue, efficient algorithms to reduce energy consumption and to guarantee quality of service (QoS) are needed [85].

In order to provide an energy-performance trade-off, authors in paper [86] have proposed a heuristic technique. It maximizes the sum of the Euclidean distances of the current allocations to the optimal point at each server. In case of a lower resource utilization, the idle power is not effectively amortized and hence the energy per transaction will be high. On the other hand, high resource utilization leads to performance degradation which causes high energy consumption due to longer execution time.

Kim et al. [87] have proposed an algorithm for reducing power consumption in VM allocation process. They modeled a real-time service as a real-time virtual machine request based on Dynamic Voltage and Frequency Scaling (DVFS) algorithm. They claimed that their algorithm could reduce the power consumption, which in turn decrease the electricity cost. However, contrary to us, they have not considered the SLA violation.

Authors in paper [88] found that using proper resouce allocation policy can significantly reduce the total energy and financial costs. They attempted to analyze how various allocation policies affect energy consumption as well as CPU load on a real cloud environment based on dynamic website loads. They defined an architectural framework for energy-efficient cloud computing. The proposed energy-aware allocation policy provides resources to cloud clients in a way that it reduces energy consumption of the datacenter while delivers the negotiated Quality of Service (QoS). The experiment results have demonstrated that the proposed model offers significant cost savings and demonstrates high potential for the improvement of energy efficiency.

The main drawback of all aforementioned studies is that they consider either energy consumption or SLA violation as their main objective and develop their solutions based on that. Unlike them, our goal is to reduce the number of active machines and decline

the energy consumption as well as meet the QoS requirements using novel multi-criteria algorithm. Another weakness of the aforementioned studies is the inability to handle multiple system resources other than CPU whereas our approach considers many important factors including CPU, RAM and storage capacity. In addition, the workload on the cluster machines is usually either lower or higher than their computing capability, which makes them either under-loaded or over-loaded [89]. The waste of resources due to many under-loaded machines is an important factor for service providers. It is also a concern for green cloud computing to save energy [90]. Authors in [91] have demonstrated that an important improvement can be achieved using 20% of resources less to do the same work. In other side, the over-loaded machines may cause a poor application performance and violate the Service Level Agreements (SLAs) [92]. Hence, efficient resource allocation strategies are needed to maintain the load well-balanced among the cluster machines as well as to meet the QoS requirements of cloud clients.

To deal with the aforementioned problems, we proposed an algorithm that efficiently assigns tasks to the set of machines (PMs) to minimize the overhead of task migration and fully use the available resources [93]. In other words, the main idea of our approach is to develop a new load balancing algorithm that manages assigning tasks to physical machines (PMs) in optimal manner, then reduces energy consumption by the set of PMs using *Matching Game theory* [94].

The rest of the chapter is organized as follows: Section 4.2 introduces a backgound on matching game theory. Section 4.3 presents the architectural framework and system model used in this chpater. Section 4.4 presents the problem formulation and our objective function. Section 4.5 proposes a load balancing technique for tasks scheduling towards resources optimization in cloud computing. Section 4.6 details the performance evaluation of our approach. Section 4.7 concludes this chapter by highlighting our contributions.

4.2 Background on matching game theory

Generally in game theory, there are three types of matching problems: one-to-one matching problem (the stable marriage problem), many-to-one matching problem (the college admission problem), and many-to-many matching problem (the firm worker problem). In this chapter, we are interested by the many-to-one matching problem, but we will present, first, a variant of one-to-one matching problem as necessary background for our

idea.

The Stable Marriage problem (SM) was introduced and studied by Gale and Shapley [95]. In the marriage model there are two finite and disjoint sets M and W, let them be men and women. $M = \{m_1, m_2, \ldots, m_n\}$ is the set of men. $W = \{w_1, w_2, \ldots, w_p\}$ is the set of women. Together they form a set of actors $A = M \cup W$. Each man has a complete preference ordering over the set $W \cup \{u\}$, and each woman has a complete preference ordering over $M \cup \{u\}$, where u denotes the possibility of remaining unmarried. The goal is to match the men with the women so that there are no two people of opposite sex who would both rather marry each other than their current partners. Preferences can be represented by a rank order lists of the form $P(m_i) = \{w_5, w_3, \ldots, m_i\}$, meaning that man m_i favors w_5 the most as its partner, w_3 the next $[w_5 >_{m_i} w_3]$ and so on, until at some point he prefers to be unmatched (i.e. matched to himself). The symbols $>_{m_i}$ and $>_{w_i}$ indicate the preference orderings of man m_i and women w_i, respectively. An agent's preferences are called strict if he or she is not indifferent between any two distinct potential assignments [96].

Definition 4.2.1 *An outcome of the marriage problem is defined by a function $\mu : M \cup W \rightarrow M \cup W \cup \{u\}$, such that $w = \mu(m)$ if and only if $\mu(w) = m$. That is, an outcome matches agents on one side to agents on the other side, or to themselves, and if w is matched to m, then m is matched to w. Agents' preferences over outcomes are determined solely by their preferences for their own mates at those outcomes [97].*

Definition 4.2.2 ***(Blocking pair)**: Given a marriage P, a pair (m, w), where m is a man and w is a woman, is a blocking pair if and only if m and w are not partners in P, but m prefers w to $M(m)$ and w prefers m to $M(w)$.*

Definition 4.2.3 ***(Stable Marriage)**: Marriage P is stable if and only if it has no blocking pairs. In the marriage model, a stable matching is efficient and the set of (pairwise) stable matchings equals the core of the game whose rules can be matched if and only if both agents from opposite sides agree.*

Theorem 4.2.1 *For every marriage problem, there exists a stable matching.*

Proof 1 *The proof of this theorem was constructed by means of a Deferred Acceptance Algorithm [95].*

4.3 Green cloud architecture

4.3.1 Architectural framework

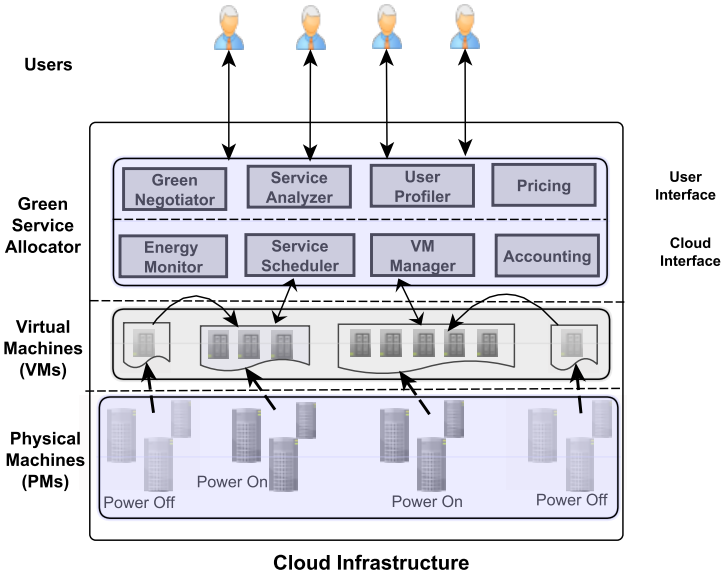


Figure 4.1 – *Green cloud architecture [98]*

A datacenter, which is home to the computation power and storage, is central to cloud computing and contains thousands of devices like servers, which are interconnected through a network with a number of switches and routers [99]. Cloud providers aim to drive the design of the next generation datacenters by collecting them as networks of virtual services, so that clients can access and deploy their applications from anywhere in the world on demand at competitive costs [100]. We assume that the network infrastructure provides enough bandwidth to avoid queueing delays in the intermediate network nodes. In cloud computing, resources are shared among a large number of tenants through the datacenter network. Each tenant may run multiple applications, thus requesting a large amount of resources. Therefore, the number of servers leased by each tenant is large. Figure 4.1 shows the high-level architecture for supporting energy-efficient service allocation in a green cloud computing infrastructure [98] considered in this paper. The architecture contains basically four main entities:

- Users: Users submit service requests from anywhere in the world to the cloud datacenter to be processed.

- Green Service Allocator: Acts as an interface between cloud infrastructure and users.
- Virtual Machines: In order to meet accepted requests, multiple VMs can be dynamically started and stopped on a single physical machine.
- Physical Machines: The datacenter comprises multiple computing servers that provide resources to meet service demands.

Resource allocation in cloud datacenters is a complex process that requires matching of a large amount of software and hardware with a large number of requests without violating the SLA. In this paper, we consider the creation and allocation of virtual machines in cloud datacenter as a part of the task, requiring a well-defined amount of resources such as CPU, memory, storage, etc. from the datacenter.

4.3.2 System model

In this chapter, the target system is an IaaS environment represented by a large-scale datacenter consisting of N heterogeneous hosts or physical machines, where each host, denoted as p, is characterized by the following parameters:

- N_CPU (p): Number of physical cores of p.
- P_CPU (p) : CPU performance of p (in Million Instruction per second MIPS).
- RAM (p) : RAM capacity of p.
- SC (p) :Storage capacity of p.
- Cost(p) : Cost per hour of p.
- OS (p): Type of the operating system running on p.
- HC (p): Has specific characteristics (like GPU ...).

Each physical machine p hosts a set of VMs with the corresponding virtual machine monitor (VMM). The percentage of CPU utilization is used to judge whether a VM has enough resources available for a service.

Multiple independent users submit requests for provisioning of M heterogeneous VMs to execute their tasks which are characterized by:

- RAM (t) : Amount of RAM required by task t.
- R_PP (t): Required processing power by task t defined in MIPS.

- DS (t) : Disk storage required by task t.
- OS (t): Type of the operating system required by task t.
- NC (t) : Need a specific characteristics (like GPU, ...).

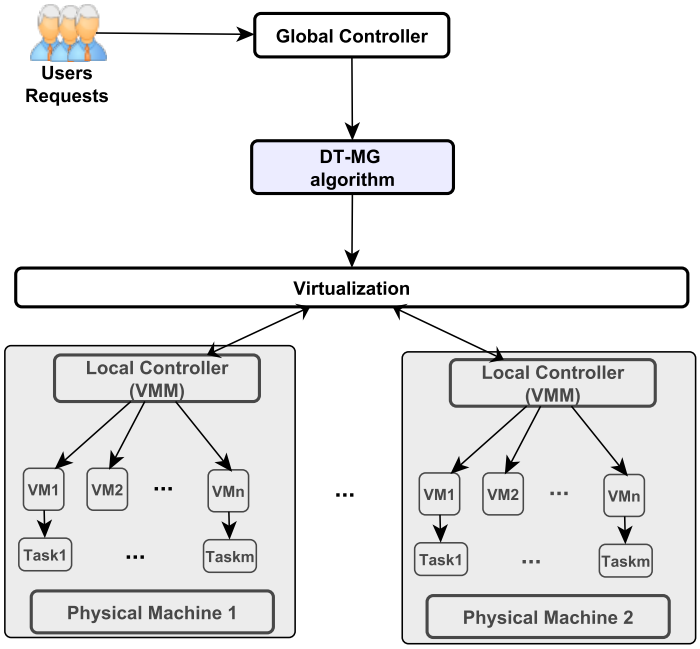


Figure 4.2 – *System model [101]*

The proposed system encompasses two controllers; Global Controller (GC) and Local Controller (LC) as shown in Figure 4.2, which is a modified version of the model described in [101]. LC acts as Virtual Machine Manager (VMM) for each PM and monitors resource (CPU) utilization continuously. Furthermore, it identifies the under-loaded and ove-rloaded PMs during execution based on our algorithm (DT-MG) as well as decides when and which tasks have to be migrated. On the other hand, GC resides on a master host and retrieves information from LC to maintain the overall view of the resources utilization and to enable dynamic scheduling. Based on the decision of the LC, the GC generates migration command to perform task placement.

4.4 Task migration and machine selection problem

4.4.1 Problem formulation

The key challenges that we address in this chapter are:

(i) How to efficiently schedule tasks on machines (PMs) in order to meet cloud clients requirements?

(ii) How to optimally solve the trade-off between resource savings and delivered performance?

(iii) How to determine when, which, and where to migrate tasks in order to minimize energy consumption as well as minimizing the migration overhead and ensuring SLA?

Therefore, to formally model the studied problem, we propose a novel approach based on a framework of *College Admissions Game*, also known as *Many-to-one Matching Game*. The College Admissions problem, as introduced in [102], models the interactions between a set of students wishing to apply to a set of colleges. Each of these colleges has a fixed quota on the number of students that it can admit. Considering the likely conflicting preferences of the students, it is worth investigating how to efficiently assign students to colleges, satisfying as best as possible their respective preferences. The Matching game theory is a promising approach to implement tasks scheduling in cloud computing environments [103]. For our purpose of designing a tasks scheduling method in cloud computing context, we formulate a college admissions game, referred to as task scheduling admissions game, which is defined by the following three components:

1. The set $T = \{t_1, t_2, \cdots, t_N\}$ of tasks acting as students.
2. The set $D = \{p_1, p_2, \cdots, p_M\}$ of physical machnies acting as colleges, each p having a certain quota q_p on the maximum number of tasks that it can simultaneously execute.
3. Preference relations for tasks and physical machines allowing them to build preferences over one another.

4.4.2 Objective function and feasibility constrains

Our aim is to reduce energy consumption in a cloud environment. To achieve this, we will efficiently schedule tasks on physical machines (PMs) to minimize the number of tasks migration and to fully use the available resources. We will then reallocate tasks on PMs and spread the load over the available machines in such a way that under-loaded PMs can be used to host some tasks of the over-loaded ones. At the same time, our approach aims to maximize the use of the cloud resources by freeing up resources that

can sit idling and yet drawing power and putting the under-loaded PMs into some form of sleep/power-saving mode while maximizing the utilization of the used ones. The PMs that are powered down when not used will be powered back on only when it is absolutely necessary. Maximizing resource utilization provides various benefits such as the rationalization of maintenance, IT service customization, QoS and reliable services. Therefore, we will assign each task to PM on which resource consumption for executing the task is explicitly or implicitly minimized without violating contractual Service Level Agreements (SLAs). SLAs define the negotiated agreements between service providers and consumers and include Quality of Service (QoS) parameters, such as task execution time. This implies that there is a trade-off between the quality of task scheduling and the resource consumption. The main objective of our approach is to deal with this trade-off by balancing these two performance metrics. In other words, we will minimize resource consumption as low as possible while meeting QoS requirements.

A cloud provider has to consider user requirements and datacenters characteristics to look for the best tasks allocation. The problem of minimizing the energy consumption of a datacenter (D) can be formulated as:

$$Minimize \quad EC(D) \tag{4.1}$$

Subject to:

$$\sum_{t_i \in d_j} RAM(t_i) \leq RAM(p_j) \tag{4.2}$$

$$\sum_{t_i \in p_j} (SI(t_i) + OI(t_i)) \leq SC(p_j) \tag{4.3}$$

The associated constraints are given in Eq. 4.2 and 4.3. Constraint 4.2 ensures that the total amount of RAM required by all tasks running on datacenter p_j does not exceed its memory capacity. Constraint 4.3 guarantees that the total disk space needed by all tasks assigned to p_j should not exceed its storage capacity.

4.5 Energy-aware allocation using Game Theory

In this section, we propose a load balancing algorithm named DT-MG. The main idea of this algorithm is to set up a *lower threshold* and an *upper threshold* and keep the total CPU utilization in-between. When the load on a machine is below (resp. above) the

lower (resp. upper) threshold, then the load is unbalanced and therefore some tasks have to be migrated. When the use of CPU goes below the lower threshold on a given PM, then all its tasks have to be migrated and afterwards the PM is disabled to save idle resource consumption. This is a kind of dynamic resources consolidation. On the other hand, if the actual CPU load goes beyond the upper threshold, which is the maximum CPU use rate, some tasks have to be migrated to avoid SLA violation.

Our approach is consisted of four procedures as shown in Figure 4.3 , which are explained in more details in the next sections.:

1. Designing the basic task scheduling.
2. Detecting under-loaded/over-loaded physical machines.
3. Selecting tasks to migrate.
4. Choosing the best migration targets.

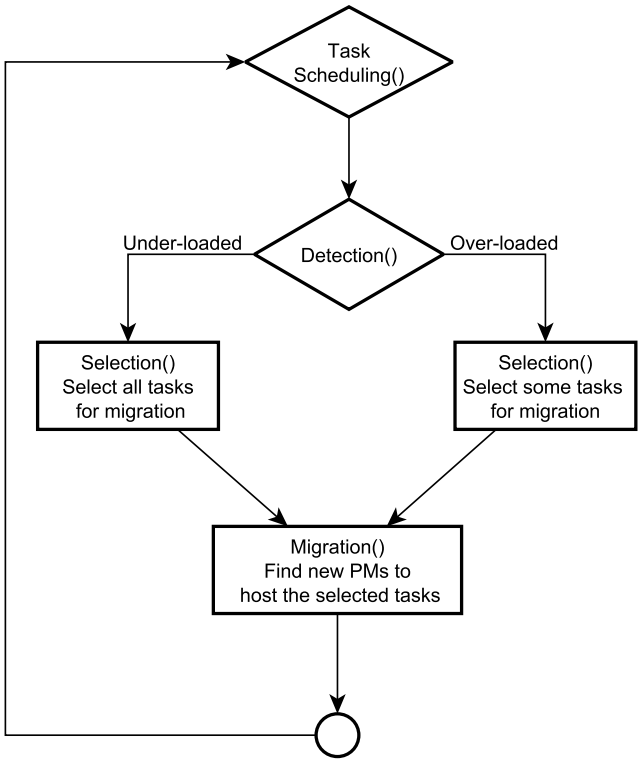


Figure 4.3 – *DT-MG mechanism running on all physical machines.*

4.5.1 Tasks scheduling procedure

4.5.1.1 Task Scheduling as a College Admissions Problem

We define a finite set of tasks $T = \{t_1, t_2, \cdots, t_N\}$ and a finite set of physical machines $D = \{p_1, p_2, \cdots, p_M\}$. Each task $t \in T$ induces a transitive *preference relation* denoted as $>_t$ over D, i.e $p_1 >_t p_2$ means d_1 is preferred to p_2 for the execution of task t. Each physical machine $p \in D$ induces a transitive *preference relation* denoted as $>_p$ over $T \cup \{\emptyset\}$, i.e $t_1 >_p t_2$ means t_1 is preferred to t_2 on p. By convention, $\emptyset >_p t$ means p prefers doing nothing than running t. In addition, the capacity of each physical machine $p \in D$, which is denoted as $q_p \in \mathbb{Z}^+$, is the maximum number of tasks that it can execute concomitantly. We assume that $\sum_p q_p \geq |T|$, so that every task can expect to have one match. The solution is a stable many-to-one matching between physical machines and tasks that satisfies the quota constraint and the preference relations.

Definition 4.5.1 ***(Many-to-One Matching):*** *Let T be set of tasks and D a set of physical machines. A many-to-one matching from T to D is a function μ, $\mu : T \cup D \to D \cup \mathcal{P}(T)$, where $\mathcal{P}(T)$ is the set of subsets of T, specified as follows:*

1. *(*task allocation*):* $\forall t \in T,\ \mu(t) = p$ *means task t is allocated to the physical machine p.*
2. *(*alternative allocations*):* $\forall p \in D,\ \mu(p) \in \mathcal{P}(T)$.
3. *(*quota constraint*):* $\forall p \in D,\ |\mu^{-1}(p)| \leq q_p$.
4. *(*matching constraint*):* $t \in \mu(\mu(t))$

The matching μ indicates for each task the physical machine on which it will be launched. For a given physical machine, it provides the set of tasks that can execute (will be used for tasks migration).

Definition 4.5.2 ***(Stable matching):*** *A given matching μ is stable if the following rules are satisfied:*

1. $\nexists t \in T$ *such that* $\emptyset >_{\mu(t)} t$.
2. $\forall t \in T, \nexists p \in D$ *such that* $(t \in \mu(p)) \wedge (p >_t \mu(t)) \wedge (|\mu^{-1}(p)| \leq q_p - 1)$.
3. $\forall t_1, t_2 \in T, \forall p \in D$, *if* $(t_1 >_p t_2) \wedge (\mu(t_2) = p))$ *then* $\mu(t_1) = p$

The first rule is obvious. The second rule says that for a given task, there shouldn't be

an under-loaded eligible physical machine with a better preference than the chosen one. The third rules states that for a given task, any other task that is preferred to him on its allocated physical machine should run there too.

4.5.1.2 Preference lists

To assign each task to an appropriate physical machine, we rely on the preference lists of both tasks and physical machines.

Preference lists for the tasks:

This phase consists in creating a preference lists of physical machines $PL(t_i)$ for each task t_i on T. This list contains the set of potentially interesting physical machines which are put in an ascending order according to their processing power, cost per unit of time, memory and storage capacities. So, physical machine having the best values for these parameters depending on the user requirements gets the highest rank.

We can calculate the processing power (PP) [101] of each physical machine using formula (4.4).

$$PP(p_j) = N\_CPU(p_j) * P\_CPU(p_j) \tag{4.4}$$

Where $N\_CPU(d_j)$ is the number of cores of physical machine p_j, and $P\_CPU(p_j)$ is the CPU performance of each core in p_j (in Million Instruction per second MIPS).
Based on formula (4.4) we can calculate the expected execution time of a given task t_i on a given physical machine p_j using equation (4.5) :

$$ET(t_i) = \frac{NI(t_i)}{PP(p_j)} \tag{4.5}$$

Where $NI(t_i)$ is the number of instructions of task t_i (in Millions of instructions MI).
The processing cost (PC) of a task t_i executed on physical machine p_j can be calculated using formula (4.6):

$$PC(t_i) = Cost(p_j) * ET(t_i) \tag{4.6}$$

Where $Cost(p_j)$ is the cost per hour for p_j.

So, each task builds its preference list $PL(t_i)$ based on its compatibility with the features of the physical machine. The list is sorted by the processing power (PP), the cost

per hour, etc. For a task t_i, a physical machine p_j will be eligible if the following constraints are verified:

$$RAM(p_j) \geq RAM(t_i) \tag{4.7}$$

$$SC(p_j) \geq DS(t_i) \tag{4.8}$$

$$NC(t_i) \leq HC(p_j) \tag{4.9}$$

$$ET(t_i) \leq DET(t_i) \tag{4.10}$$

$$PC(t_i) \leq Budget(t_i) \tag{4.11}$$

Eq. (4.10) means that the execution time of a task t_i on a physical machine p_j should be less than the desired execution time of the task t_i. Eq. (4.11) means that the processing cost of a task t_i on a physical machine p_j should be less than the cloud client's budget dedicated for this task.

Preference lists for the physical machines:

The preference list of physical machine p_j is a set of tasks $PL(p_j)$, where each task has a load that does not exceeds its capacity. These tasks are sorted in ascending orders according to different parameters. In the proposed strategy, the tasks are initially prioritized according to their impact on the use of the physical machine in terms of CPU, RAM, etc. Thus, the key factor for prioritizing tasks is their CPU utilization ratio.

Each physical machine selected to run a task t_i calculates its CPU utilization ratio (UR) [104] using formula (4.12).

$$UR(t_i, p_j) = \frac{CPU(t_i)}{PP(p_j)} \tag{4.12}$$

Where $CPU(t_i)$ is the quantity of MIPS required by the task t_i and $PP(p_j)$ is the processing power of a physical machine p_j. For a physical machine p_j, a task t_i will be eligible if the following constraints are verified:

$$UR(t_i, p_j) \leq 1 \tag{4.13}$$

$$DS(t_i) \leq SC(p_j) - \sum_{t_k \in p_j} DS(t_k) \tag{4.14}$$

$$RAM(t_i) \leq RAM(p_j) - \sum_{t_k \in p_j} RAM(t_k) \tag{4.15}$$

4.5.1.3 Proposed algorithm for task scheduling

So far, we have modeled our problem as a many-to-one matching game where the players are the tasks and the physical machine. Each task has the right to choose one physical machines. By contrast, the physical machines can host one or more tasks up to its quota q_p. As a solution, we present the matching algorithm as shown in algorithm 3 based on the deferred acceptance algorithm introduced in [95]. Figure 4.4 presents the

Algorithm 3: Pseudo code of Task-PM matching algorithm

```
Input: Tasks' preference lists (PLs).
       PMs' preference lists (PLs).
while (∃ a free task t_i that still has a physical machine p_j to propose to) do
    Task t_i proposes to all physical machines on its preference list.
    if (q_p ≥ 0) then
        t_i is assigned to the waiting list of p_j
        else
            Compare t_i with the current tasks on the waiting list for
            the physical machine p_j and reject the task that has least preference.
    The rejected tasks re-apply to their next best choice.
    Each physical machine update its waiting list based on its preference list, and
     rejects the rest.
Output: Every t_i is on a waiting list of one of the set of physical machine,
        and,thus, we have convergence to a stable matching.
```

flowchart of the proposed algorithm.

4.5.2 Detection of physical machines (PMs) state procedure

According to the study [105], the resource consumption of a physical machine can be divided into 6 levels, an idle state and five levels of CPU utilization at running state as shown in Figure 4.5. The CPU utilization threshold depends on hardware architecture and may differ on different cloud systems. Based on the hardware commonly found in PMs and our research model, a 70% CPU utilization threshold is considered an appropriate cutoff point [106]. For the sake of simplicity, we shall use 70% CPU utilization as the default upper threshold (UPTH) and 20% CPU utilization as the default lower threshold (LOTH) for the rest of the paper. In other words, when the CPU load of a PM is below

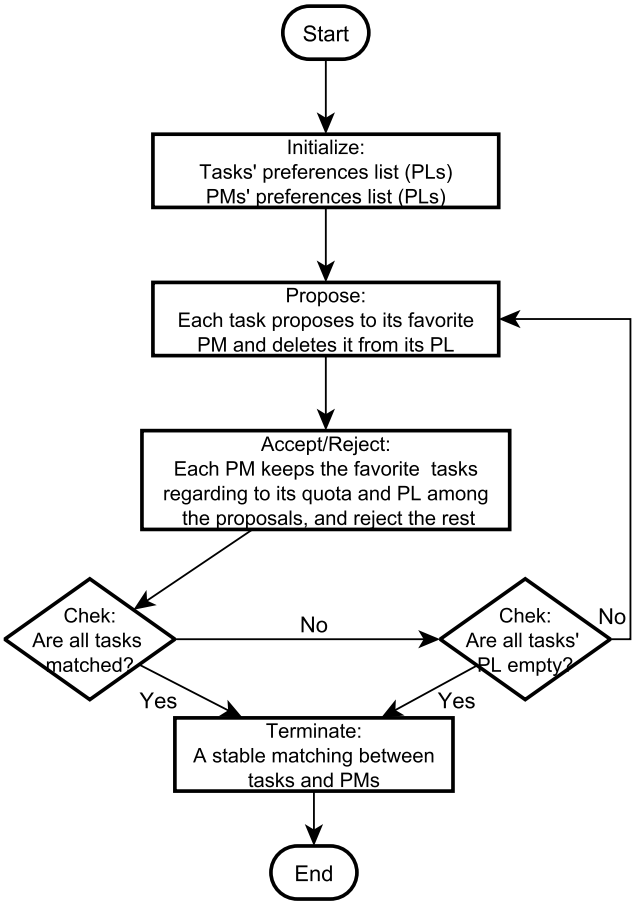


Figure 4.4 – *Flowchart of Task-PM matching algorithm*

than lower threshold (LOTH), we tend to power down this PM and migrate all its tasks to another PM with low overhead based on the preference lists as shown in section 4.5.1.2. On the other hand, when the CPU load of a PM exceeds the upper threshold (UPTH), some tasks of this PM should be migrated to balance the load among PMs and to reduce the risk of SLA violation.

The percentage of CPU load is used to specify whether a physical machine has enough resources available to execute clients' tasks. [104].
The load of a physical machine p_j can be calculated as the total computational length of all tasks running on p_j divided by its processing power $PP(p_j)$, as shown in formula (4.16).

$$Load(p_j) = \frac{\sum_{t_i \in p_j} CL(t_i)}{PP(p_j)} \tag{4.16}$$

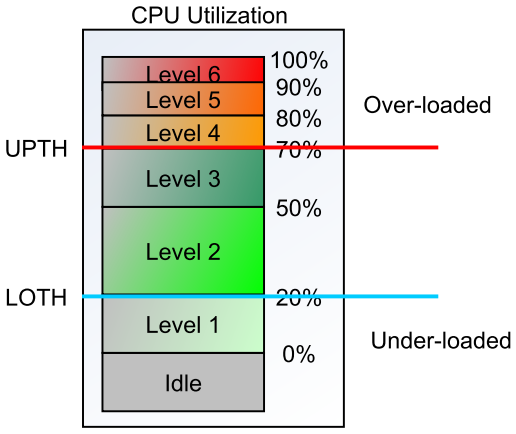


Figure 4.5 – *Levels of PM' CPU utilization [105]*

Where $CL(t_i)$ is the computational length of a task t_i (in Millions of Instructions MI). By calculating $Load(p_j)$, we can detect if the state of p_j is over-loaded or under-loaded.

4.5.3 Tasks selection procedure

The question that naturally arises is when a task migration should take place. So, as mentioned before and based on load percentage, a migration may be initiated whenever a physical machine detects that it is:

- Over-loaded compared to upper threshold (UPTH), i.e. the CPU load of some physical machines is close to 100% that creates a risk of SLA violation.
- Under-loaded compared to lower threshold (LOTH), which is not efficient in terms of resource consumption.

As aforementioned if the CPU load of a physical machine falls below the lower threshold(LOTH) we migrate all tasks to another machine and then switch it to low-power mode (i.e. sleep, hibernation) in order to eliminate the idle power consumption. On the other hand, if the CPU load exceeds the upper threshold (UPTH), some tasks have to be migrated from this machine to balance the load among physical machines.

Algorithm 4 presents our pseudo-code for tasks selection procedure running on over-loaded physical machines (PMs). It sorts the list of tasks in decreasing order of the CPU utilization. Then, it repeatedly looks through the list of tasks and finds the best task to migrate from the over-loaded PM. The best task is the one that satisfies two conditions:

First, by migrating it, the load of the PM falls to a new value below upper threshold (UPTH). Second, the difference between the upper threshold(UPTH) and the new CPU load percentage is the minimum across the values provided by all tasks on this PM. If there is no such task, the algorithm selects the task with the highest computational length, adds it to the list of tasks to be migrated, and removes it from the initiated list of tasks. The algorithm continues until the load of the physical machine goes below UPTH.

Algorithm 4: Tasks Selection() Procedure runs on over-loaded physical machines

```
   Input: D: set of physical machines
   Output: migrationList
1  taskList ← Sorted list of tasks in descending order based on computational
     length.
2  while (load(p_j) > UPTH) do
3      Max ← (UPTH − LOTH) ∗ α;   0 < α < 1
4      bestDiff ← Max
5      for each t_i in taskList do
6          load'(p_j) ← load(p_j) − load(t_i)
7          if load'(p_j) < UPTH then
8              if bestDiff > UPTH − load'(p_j) then
9                  bestTasks ← t_i
10                 bestDiff ← UPTH − load'(p_j)
11                 else if (bestDiff = Max) then
12                     bestTasks ← t_i
13             break
14     load(p_j) ← load'(p_j)
15     Add bestTasks to migrationList
16     Remove selected tasks from taskList
17 return migrationList.
```

4.5.4 Tasks migration procedure

Target physical machine selection is the most critical step in task migration phase. Wrong physical machine selection may increase the number of task migration as well as the resource wastage [107].So, for each active task to be migrated, our approach calculates the expected execution time and the task migration cost on each destination machine in its preference list. Then, it picks the one of maximum gain. We calculate the expected execution time of each migrated task on each machine p_j in its preference list using formula

(4.5) and the task migration cost using (4.17) :

$$MC(t_i) = Cost(p_j) * ET_{p_j}(t_i) + C_{BW} * CL(t_i) \tag{4.17}$$

Where $MC(t_i)$ is the migration cost of task t_i. $Cost(p_j)$ is the cost per hour for the destination PM p_j. $ET_{p_j}(t_i)$ is the expected execution time for the task t_i on p_j. C_{BW} is bandwidth cost among the hosted machine and destination machine and $CL(t_i)$ is the computational length of a task t_i (in Millions of Instructions -MI).

If there are multiple physical machines available for receiving the migrated tasks in its preference list, the one with minimal migration cost and minimal execution time will be chosen to host the task. Algorithm 5 presents the pseudo code of task migration procedure. It tries to find best physical machines with sufficient capacity to host the migrated tasks.

Algorithm 5: Task migration procedure

```
1  /* Running on under-loaded PMs. */
2  for each t_j in under-loaded node p_j do
3      for each p_j in preference_list of a task t_i do
4          if load(p_j) + load(t_i) < UPTH then
5              Migrate t_i to the node p_j
6              load'(p_j) ← load(p_j) + load(t_i)
7  Turn of p_j
8
9  /* Running on over-loaded PMs. */
10 Tasks Selection Procedure(),
11 for each t_j in migrationList do
12     for each p_j in preference_list of a task t_i do
13         if load(p_j) + load(t_i) < UPTH then
14             Migrate t_i to the node p_j
15             load'(p_j) ← load(p_j) + load(t_i)
16             break
17 /* exit procedure */
```

4.5.5 Convergence of our approach

In this chapter, we have used *matching game theory* to design and analyze a distributed load balancing (DT-MG) mechanism for energy-efficiency in cloud datacenter. This matching problem with aggregating stated preferences becomes a non-cooperative game between the set of physical machines (PMs). In this section we demonstrate that our proposed approach (Tasks Migration Procedure) always converges to a pure Nash equilibrium.

Nash equilibrium is the most important concept in game theory, which is a static stable strategy vector that has chosen by each player in the game and no player can benefit by unilaterally changing his or her strategy while the other players' strategies remain unchanged [108].

Definition 4.5.3 *The DT-MG reaches a Nash equilibrium when none of the migration procedures were able to unilaterally find a better PM for their extra load.*

Definition 4.5.4 ***(potential function):*** *We define the potential function F of datacenter at a given time using the number of well-loaded PMs and the number of active PMs [109], which equals:*

$$F \triangleq \frac{nbr\,of\,well^{-}loaded\,PMs}{nbr\,of\,active\,PMs} \tag{4.18}$$

We demonstrate that the potential function F is strictly increasing with a maximum value and thus, the approach must consequently stop at some point.

Theorem 4.5.1 *DT-MG always admits a pure Nash equilibrium.*

Proof 2 *We assume that the potential function increases strictly after each successful migration. A migration may be initiated whenever a PM is over-loaded compared to upper threshold (UPTH) or under-loaded compared to lower threshold (LOTH). In addition, a successful migration occurs in three cases:*

- *An over-loaded physical machine (PM) finds a better machine (in the set of active PMs) to host its extra load. In this case, the state of the over-loaded PM converts to the well-loaded. The number of well-loaded PMs increments. Thus, F increases.*
- *An over-loaded physical machine (PM) is not able to find a better machine to host its extra load, it thus turns on an idle physical machine (PM). Then, the number of both well-loaded and active PMs increments. Thus, F increases.*

- *An under-loaded physical machine (PM) migrates all tasks to another PM and goes to idle state. The number of active PMs decrements, and F again increases.*

The potential function F is strictly increasing, and it has bounded to one. Therefore, the game converges after a limited number of successful steps to a pure Nash equilibrium.

In our approach, we try to find the best task t_i among all tasks on over-loaded node PM_j to migrate. The best task is the one that satisfies two conditions: First, by migrating it, the load of the PM falls to a new value below upper threshold (UPTH). Second, the difference between the upper threshold (UPTH) and the new CPU utilization percentage is the minimum across the values provided by all tasks on PM_j. If there is no such task, the algorithm selects the task with the highest computational length, adds it to the list of tasks to be migrated, and removes it from the initiated list of tasks.

For each task t_i to be migrated that already assigned to PM_j, we then try to find out a best new host. That is, among all the other nodes PMs (except PM_j itself) with enough remaining capacity to accept t_i, we select the PM that can yield the minimal migration cost and minimal execution time as the best node PM to host the migrated task.

With N tasks to be scheduled on M PMs, assuming the average number of assigned tasks on a node PM is K, then, the time complexity of algorithm 4 is $O(KM) \sim O(N)$. Considering the two loops in algorithm 5, the total time complexity is $O(N^2)$.

4.6 Performance analysis

4.6.1 Data center settings

Cloud computing is an open platform which consists of multiple distributed datacenters. A datacenter constructs all kinds of resource pools by virtualizing physical resources, and encapsulates resources into services. Cloud users submit their requests through a variety of cloud applications established by cloud service providers. Subsequently, user requests are submitted to the datacenter in the form of tasks. Then, according to the scheduling strategies and the monitored state of resources, datacenter schedules each task into the appropriate PM.

As the target system is a generic cloud computing environment, it is essential to evaluate it on a large-scale virtualized datacenter infrastructure. However, carrying out experiments on a real cloud platform would be very expensive and difficult, especially when it

is necessary to reproduce the experiment with the same conditions to compare different algorithms. Therefore, a simulation has been chosen as a way to evaluate the proposed algorithm. We used CloudSim 3.03, which is an extensible simulator which aims to enable modeling and simulation of cloud-based systems [110, 111]. In contrast to another simulation toolkits (e.g. SimGrid, GangSim), it permits the modeling of virtualized environments, supporting on demand resources provisioning and their management.

We have simulated one datacenter composed of 800 heterogeneous physical machines (PMs), half of which are HP ProLiant ML110 G5 and the other half consists of HP ProLiant ML110 G4. The characteristics of the PMs in terms of power consumption are given in Table 4.1 [112]. The PMs comprise of 2500, 2000, 1000 and 500 MIPS accordingly. There are 1000 VMs of different types on the datacenter: Medium Instance (2 600 MIPS, 0.9 GB); Extra Large Instance (2 500 MIPS, 3.8 GB); Small Instance (1 200 MIPS, 1.5 GB) and Micro Instance (250 MIPS, 600 MB). The amount of tasks in experiments is 500, 1000 and 1500.

nodes (PMs)	**0%**	**20%**	**40%**	**60%**	**80%**	**100%**
HP G4 (Watt)	86	92.6	99	106	112	117
HP G5 (Watt)	93.7	101	110	121	129	135

Table 4.1 – *Power Consumption by the Selected PMs at Different Load Levels in Watts*

As input data, we have used PlanetLab data on the CPU traces collected from more than 1000 VMs running on almost five hundred different locations around the world [113]. The data have been collected every five minutes during the period from the 03rd May to 20th of April 2011. The characteristics of the data for each day are shown in Table 4.2.

4.6.2 Performance metrics

In order to compare the efficiency of our proposed approach, we have used four metrics to evaluate its performance.

(a) Energy consumption (kWh): The first performance metric is called the total energy consumption which can be calculated based on the CPU load of each physical machine (PM) on the datacenter as shown in Table 4.1 [88] .

(b) Number of Task migration: The second metric is the total number of task migration which computes the number of migrated tasks during the execution process.

Date	Num. of VMs	Mean (%)	St. dev. (%)
03/03/2011	1052	12.31	17.09
06/03/2011	898	11.44	16.83
09/03/2011	1061	10.70	15.57
22/03/2011	1516	9.26	12.78
25/03/2011	1078	10.56	14.14
03/04/2011	1463	12.39	16.55
09/04/2011	1358	11.12	15.09
11/04/2011	1233	11.56	15.07
12/04/2011	1054	11.54	15.15
20/04/2011	1033	10.43	15.21

Table 4.2 – *Workload data characteristics (based on CPU utilization) [113]*

(c) SLA violation: Cloud service providers should ensure the satisfaction of application demands, namely meeting the requirement of client's quality of service (QoS). QoS requirements are commonly formalized in the form of SLA, which can be determined in terms of such characteristics as minimum throughput or maximum response time delivered by the deployed system [114].

The third performance metric is the percentage of SLA violation which describes how many times the allocated resources are less than the required resources. An SLA violation occurs when a given PM cannot provide the amount of Million Instructions per Second (MIPS) that is requested. We have measured the SLA violation with two aspects: i) SLA violation time per active host (SLATAH) which is the time when the CPU load of a machine is near to 100%; and ii) performance degradation due to migrations (PDM) that occurs due to tasks migration during the load balancing process or switching off under-loaded machines [115]. Those two aspects can be described in Eq. (4.19) and (4.20) [116].

$$SLATAH = \frac{1}{M}\sum_{i=1}^{M}\frac{T_{p_j}}{T'_{p_j}} \tag{4.19}$$

$$PDM = \frac{1}{Q}\sum_{j=1}^{M}\frac{C_{p_j}}{C_{t_i}} \tag{4.20}$$

Where T_{p_j} is the SLA violation time on physical machine p_j. T'_{p_j} is the active time of the physical machine p_j and M is the overall physical machines (PMs) in the datacenter (D). C_{p_j} is an estimate of performance degradation of p_j caused by migrations and C_{t_i} is the total CPU capacity requested by a task t_i during its lifetime.

There is also a combined metric that encompassed both SLATAH and PDM metrics. It is called SLAV and it is the main metric to measure SLA violation. It is calculated as (4.21).

$$SLAV = SLATAH * PDM \tag{4.21}$$

(d) Energy and Service level agreement Violation (ESV): This metric combines both energy consumption and SLA violation metrics [101]:

$$ESV = Energy * SLAV \tag{4.22}$$

We have compared our proposed algorithm (DT-MG) with seven other algorithms, which are:

(1) Non-Power Aware (NPA): It does not apply any energy optimization. In this policy all nodes operate at 100% CPU usage and consume maximum power all the time.

(2) Dynamic voltage and frequency scaling (DVFS): It is a commonly-used power-management technique where the clock frequency of a processor is decreased to allow a corresponding reduction in the supply voltage. This reduces power consumption, which can lead to significant reduction in the energy required to run a job [117].

(3) Static Threshold and Minimum Migration Time (ThrMmt): It uses a static Upper Threshold for detecting over-loaded nodes and *Minimum Migration Time* for selecting tasks to be migrated. [101].

(4) Median Absolute Deviation and Minimum Migration Time (MadMmt): In this technique the overload threshold is calculated dynamically using *median absolute deviation* and, then, it selects a task to migrate which has the least computational length as it will be migrated faster [118].

(5) Median Absolute Deviation and Random Selection (MadRs): This technique uses the *median absolute deviation* to detect the over-loaded PMs and migrates tasks randomly without applying any rules [101].

(6) Utilization and Minimum Correlation (UMC): It takes both machine utilization and machine correlation with VM for choosing a suitable machine to host the migrated tasks. They have also proposed *VM-based dynamic threshold (VDT)* algorithm for detecting under-loaded hosts and *Local Regression (LR)* technique for detecting the over-loaded hosts [119].

(7) Modified Best Fit Decreasing (MBFD): It presents an energy-aware resource allocation policy for efficient datacenters management in cloud computing. It used a static upper and lower thresholds for detecting the over-loaded and under-loaded hosts. Then,

a *Minimization of Migration policy (MM)* is used to migrate the minimum number of tasks. After that, a *Modified Best Fit Decreasing (MBFD)* algorithm is applied to assign each migrated task to a host that gives the least increase of power consumption [88].

For ThrMmt, MadMmt and MadRs benchmarks, they use the Power Aware Best Fit Decreasing (PABFD) technique to select the best PMs to host the migrated tasks. PABFD [101] prepares first a list of migrated tasks. Then, it sorts the migrated tasks in descending order in terms of their CPU requirements. Finally, it allocates each migrated task to a host providing the least increase of energy consumption without exceeding the upper threshold.

4.6.3 Simulation Results

First of all, we have tested our approach by calculating the number of physical machines (PMs) switched off as shown in Table 4.3. The energy consumption of a datacenter is mainly caused by running PMs, thus our approach try to minimize the number of PMs which are in use in order to reduce the energy consumption. The simulation was done on 80 PMs with different number of cloudlets in range [500-3000]. The results show that our approach minimizes the number of PMs turned on which can lead to reduce the energy consumption.

Num of cloudlets	**Num of PMs switched off**
500	13
1000	10
1500	8
2000	6
2500	3
3000	2

Table 4.3 – *Number of PMs switched off.*

Figure 4.6 compares the energy consumption (in kWh) by different algorithms disscused in this paper. As depicted in Figure 4.6, the energy consumption by our approach DT-MG is much less in comparison with other techniques whereas the NPA technique consumes the most. More precisely, our approach (DT-MG) leads to 50% reductions in energy consumption compared to the NPA technique and almost up to 20% on average compared to other techniques with different numbers of submitted tasks (cloudlets).

Figure 4.7 shows the number of task migration during the simulation, while Figure

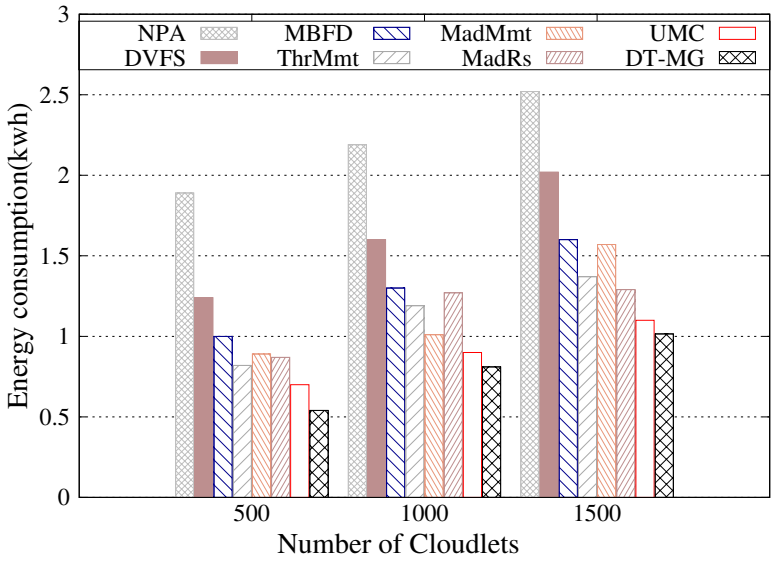


Figure 4.6 – *Energy consumption*

4.8 shows the percentage of performance degradation due to task migration.

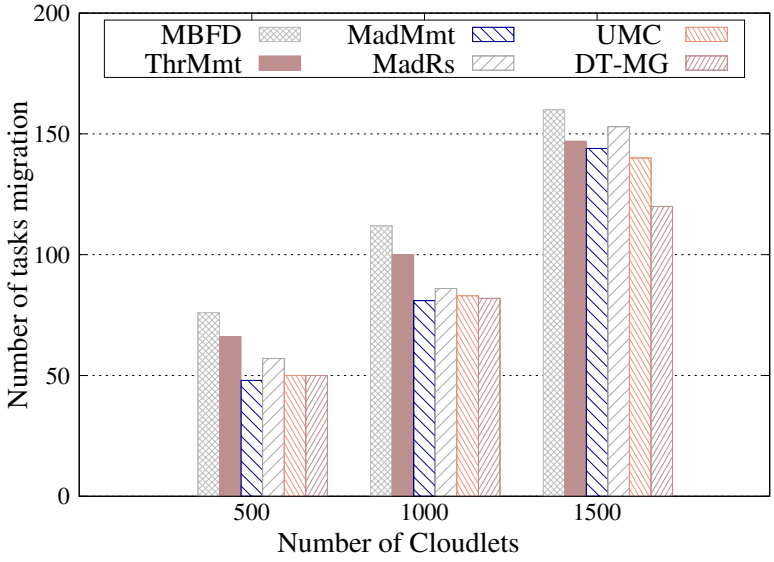


Figure 4.7 – *Number of task migration*

It is clearly visible from Figure 4.7 that our approach DT-MG performs greatly better than other techniques, which only migrates 120 times on average for 1500 tasks compared to 140 times done by UMC (the best one of the five techniques). Figure 4.8 shows that our approach DT-MG and MadMmt technique have the smallest rate of performance degradation due to task migration, mainly when the number of tasks is large. The reason of this behavior is that when the number of tasks is small our approach consolidates the load

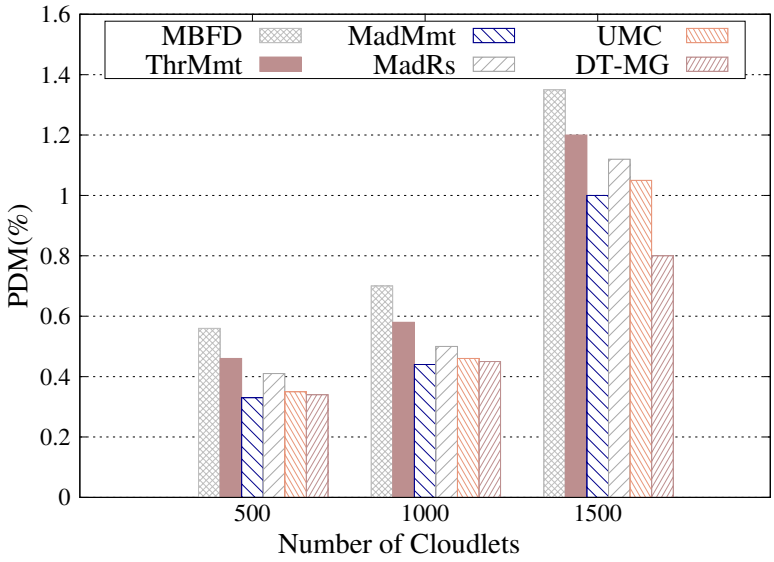


Figure 4.8 – *Performance Degradation due to Migration*

of PMs by switching off under-loaded PMs, thus, the number of task migration increases which can lead to degrade the datacenter performance. However, when the number of tasks increases, the performance degradation by our approach is reduced, because the majority of the PMs will be well-loaded. To quantify, the percentage of performance degradation due to task migration caused by our approach (DT-MG) is 0.8% with 1500 tasks, whereas the minimum of all other method is 1.01%, resulting 21% reduction.

This observation can be described by the fact that our approach allocates tasks to PMs based on both tasks and machines preference lists which can reduce the number of tasks migration. Furthermore, our approach manages efficiently the cloud resources by switching off the under-loaded machines.

The histogram obtained by plotting SLA violation time per active host (SLATAH) is shown in Figure 4.9. It affirms better performance of the proposed approach in terms of SLA violation per active host. DT-MG has smaller SLATAH compared to other techniques; therefore, the percentage of time that active PMs experienced CPU utilization near to 100% in DT-MG approach, is less with 6.72% in avergae than other techniques with different numbers of submitted tasks.

SLA violation is one of the key indicators of QoS, so, an approach having low SLA violation ensures the desired QoS. The results presented in Figure 4.10 show that our proposed approach has the small values of SLA violation which is 1.02% with 500 tasks, 2.5% for 1000 tasks and 6% with 1500 tasks, resulting 44% reduction in SLA violation on aver-

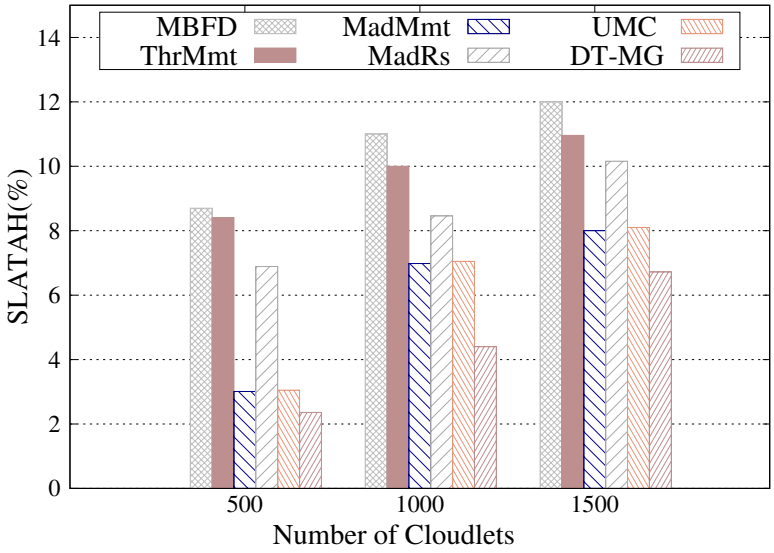


Figure 4.9 – *SLA Violation Time Per Active Host*

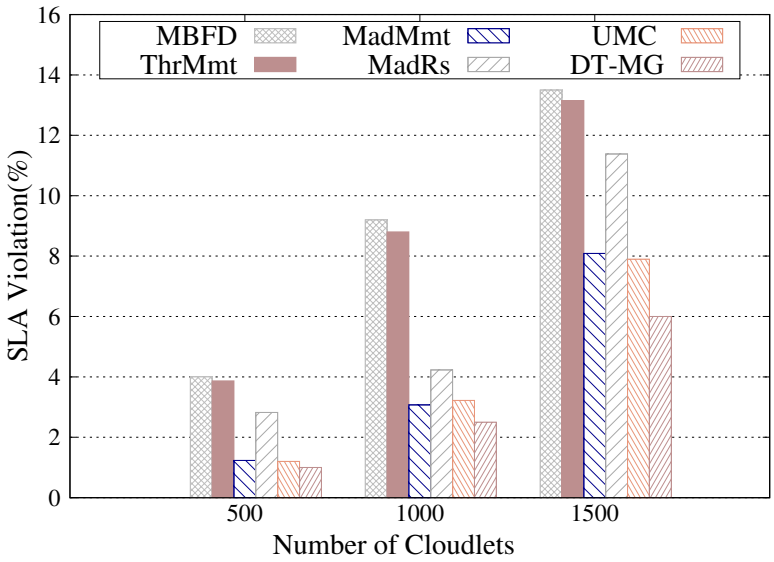


Figure 4.10 – *SLA Violation*

age with 1500 tasks. This is the main achievement of our appraoch, that is, it efficiently manages datacenter resources, and as an outcome, SLA violation reduced significantly. If tasks are assigned to the right PMs then there will be less migrations number, which will reduce SLA violation as well.

ESV is a combination of both SLAV and energy consumption metrics. It presents a tradeoff picture between our proposed approach and the other approaches discussed in this paper. Based on the results shown on Figures 4.6 and 4.10, both energy and SLA

violation were reduced, which implies that ESV will also be reduced. As a result, we have achieved the Energy and SLA trade-off. As depicted in Figure 4.11, the results for DT-MG approach regarding ESV metric is much less in comparison with MBFD, PABFD and UMC policies. More precisely, the minimum ESV value obtained by our proposed approach (DT-MG) with 1500 tasks is 70.06 while the minimum of all other approaches is 90.69 for UMC technique, which can lead to 22.75% reduction.

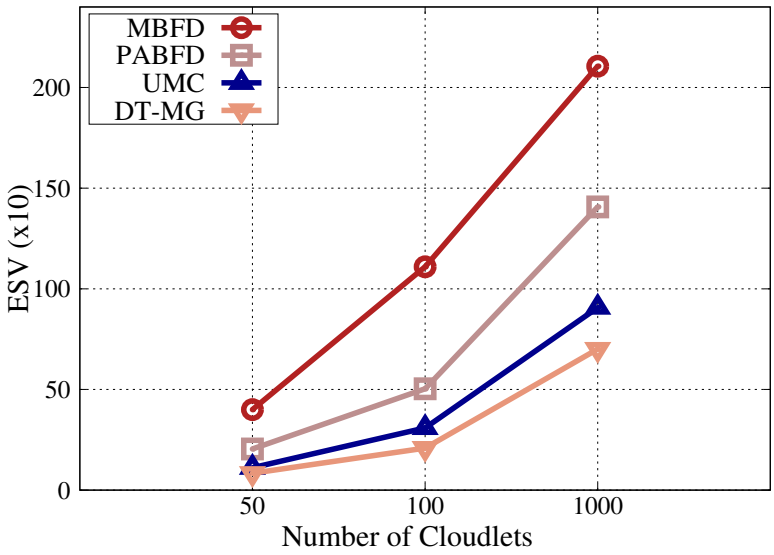


Figure 4.11 – *The ESV metric*

4.6.4 Statistical analysis

Policy	Mean of ESV($\times$10)	95% Confidence Interval	p-value
DT-MG	69.89	(69, 70.792)	$p - value > 0.05$
UMC	90.73	(90.19, 91.27)	$p - value > 0.05$
PABFD	140.32	(139.53, 141.11)	$p - value > 0.05$
MBFD	210.32	(209.77 210.86)	$p - value > 0.05$

Table 4.4 – *Comparison algorithms using one-Sample T-test regarding ESV metric*

In this section, we present a statistical analysis based on ESV metric. We have conducted One-sample T-test to compare DT-MG approach with other three techniques regarding the mean values of the ESV metric along with 95% confidence intervals (Table 4.4). From the results, all p-values are greater than the significance level 0.05 implying that the distribution of the data are not significantly different from the normal distribution. According to Shapiro-Wilk's normality test, the values of the ESV metric produced

Policy 1 (ESV $\times 10$)	Policy 2 (ESV $\times 10$)	Difference ($\times 10^{-2}$)	p-value
DT-MG (70.06)	UMC (90.69)	-0.208 (-0.225, -0.196)	$p - value < 0.001$
DT-MG (70.06)	PABFD (140.66)	-0.706 (-0.717, -0.695)	$p - value < 0.001$
DT-MG (70.06)	MBFD (210.6)	-1.401 (-1.414, -1.389)	$p - value < 0.001$
UMC (90.69)	PABFD (140.66)	-0.504 (-0.512, -0.496)	$p - value < 0.001$
UMC (90.69)	MBFD (210.6)	-1.188 (-1.197, -1.179)	$p - value < 0.001$
PABFD (140.66)	MBFD (210.6)	-0.692 (-0.705, -0.680)	$p - value < 0.001$

Table 4.5 – *Comparison algorithms using paired Samples T-test*

by the selected algorithm follow a normal distribution with the $p - value > 0.05$. From the presented results, we can conclude that our approach is much less, in terms of ESV, in comparison with MBFD, PABFD and UMC policies, which means that the Energy and SLA trade-off has been achieved. More precisely, the mean value of ESV of our approach (DT-MG) is 69.89 with 95% CI (69.005, 70.792), whereas the minimum mean value of ESV of all other policies is 90.73 for UMC policy with 95% CI (90.194, 91.271), resulting 22.97% reduction.

Table 4.5 presents the results of four paired T-test for all aforementioned policies. If the p-value is greater than 0.05, then we must accept the null hypothesis, otherwise we must reject the null hypothesis. Our null hypothesis is: "There is no significant difference in the performance between two policies regarding ESV metric". From Table 4.5 we find that the p-value is significantly smaller than 0.05 for ESV metric. Therefore, we could conclude that there is significant difference between these algorithms with $p - value < 0.001$. More precisely, the mean of the differences for DT-MG and UMC policies is -0.208 with 95% CI: (-0.225, -0.196). The mean of the differences for DT-MG and PABFD policies is -0.706 with 95% CI: (-0.717, -0.695). The mean of the differences for DT-MG and MBFD policies is -1.401 with 95% CI: (-1.414, -1.389). This means that the DT-MG policy leads to a statistically significant lower value of the ESV metric in comparison with UMC, PABFD and MBFD policies.

4.7 Conclusion

In this chapter, we have proposed a load balancing algorithm (DT-MG) for reducing the energy consumption and using efficiently the available resources while delivering the negotiated level of Quality of Service (QoS). DT-MG assigns tasks to machines using matching game theory in order to reduce the overhead of task migrations. Then, it de-

fines an upper and lower utilization thresholds and maintains the total CPU utilization of the machines between these thresholds. The experiment results have shown that our proposed algorithm (DT-MG) outperforms existing energy-aware algorithms discussed in this chapter. Our algorithm reduced the energy consumption by 50%, 40% and 20% when compared to NPA, DVFS and other techniques, respectively. It also showed a 44% reduction in SLA vioaltion on average compared to the state of the art algorithms.

As we have said before, an efficient task scheduling is critical for achieving high performance in cloud computing environment. In addition, an optimal scheduling of tasks in cloud computing is a difficult optimization problem, and many algorithms have been proposed to solve it. Among the scheduling algorithms for heterogeneous computing environments, the Heterogeneous Earliest Finish Time (HEFT) algorithm has been shown to produce shorter execution time more often than other comparable algorithms. However, the HEFT algorithm does not deal with the load balancing among the set of machines. Therefore, the key idea of the next chapter is to enhance the process of scheduling tasks in HEFT algorithm, by applying the matching game model used in this chapter to achieve a well-balanced load across the machines as well as to minimize the makespan of a given workflow application.

Chapter 5

A Static Heuristic Scheduling for Minimum Makespan

Contents

In this chapter, we propose an enhancement of Heterogeneous Earliest Finish Time (E-HEFT) algorithm under a user-specified financial constraint to achieve a well-balanced load across the virtual machines while minimizing the makespan of a given workflow application. In order to evaluate the performance of the algorithm, we compare it with some other existing scheduling

algorithms. Experimental results show that our algorithm outperforms the selected algorithms on reducing the makespan and improving the load balance among virtual machines. This chapter is derived from:

– **Yassir Samadi**, Mostapha Zbakh and Claude Tadonki. "E-HEFT: Enhancement Heterogeneous Earliest Finish Time algorithm for Task Scheduling based on Load Balancing in Cloud Computing". To appear in the 2018 International Conference on High Performance Computing & Simulation (HPCS 2018). IEEE. Orléans, France.

5.1 Introduction

Cloud clients are consuming and producing a huge volume of data that should be analyzed and processed, thus being considered as data-intensive experiments. This large volume of data can be found in many areas like astronomy [120], high-energy physics [121] and bio-informatics [122], etc. Hence, scientists need to analyze terabytes of data either from existing data resources or collected from physical devices. In order to manage the execution of these complex experiments, scientific workflows can be a prominent solution. A scientific workflow [123] describes the automation of scientific or general process and the set of rules (dependencies). In other words, scientific workflows are sets of elementary tasks and their dependencies. In addition, a given client is interested in reducing the makespan. So, the scheduler should efficiently fully utilize the available resources and make sure that the load is globally well balanced.

In a scheduling algorithm, scientific workflow can be viewed as a directed acyclic graph (DAG), where nodes (or tasks) represent the computation and edges represent the communication between them. The scheduler assigns a weight for each node in the DAG, which represents the computation cost, and assigns weights for edges corresponding to communication cost between nodes. In addition, scientific workflow applications have many computations and tasks that generate many intermediate large datasets with dependencies among them. So, the scheduler should also take care of precedence constraints between the set of tasks. As a result, over several years, a number of heuristic algorithms suitable for DAG scheduling on heterogeneous resources have been suggested [124] that attempt to strike a good balance between running time, complexity and schedule quality [125], but still a lot of work needs to be done to make scheduling more effective. The Heterogeneous Earliest Finish Time heuristic (HEFT) [42] has been one of the most often cited and used, with the advantage of simplicity and generating generally

good schedules with a short makespan. However, HEFT algorithm lacks load balancing among the machines of the cluster. In this chapter, we extend our previous works presented in chapters 3 and 4, and propose an enhancement of HEFT algorithm. In other words, our scheduling algorithm aims to minimize the total execution time of tasks as well as to achieve a well-balanced load across all VMs in cloud environment, always obeying both budget and precedence constraints imposed by the DAG. That is, there are two objectives considered here. The first one is the minimization of the tasks execution time. The second one is to evenly distribute the workload among the virtual machines of the entire cluster.

In order to reduce the total execution time of a workflow, we propose a scheduling approach that takes into account the variety and heterogeneity of virtual machines in a cloud computing cluster (e.g. different bandwidths, transfer rates, and processing capacities) [126]. It also takes into account the data distribution and data constraints all together within the same solution, i.e. tasks and data transfers are scheduled together by the E-HEFT. Thus, the scheduler defines the distribution of tasks and data among the virtual machines so as to minimize the total execution time and data transfers. Then, our scheduling algorithm is simulated using the CloudSim simulator [111, 127]. To evaluate the performance of the enhanced algorithm, a comparative study is done with some state-of-the-art algorithms.

The rest of the chapter starts with a review of heuristic algorithms for DAG scheduling in section 5.2. In section 5.3, we describe the mathematical formulation of the studied problem and our objective function. Section 5.4 presents in details the steps of our proposed algorithm. Section 5.5 assesses the applicability of our proposed solution and illustrates its performance using Cloudsim simulator. Section 5.6 outlines the conclusion.

5.2 Heuristic algorithms for DAG scheduling

5.2.1 State-of-the-art

In this section, we discuss different works related to task scheduling in cloud computing. Tasks scheduling or resources mapping for workflows has been investigated extensively in the literature in the past decade [128].

Many task scheduling algorithms have been proposed to minimize the workflow

makespan in heterogeneous environment such as Opportunistic Load Balancing (OLB) [129], Minimum Execution Time (MET) [130] and Heterogeneous Earliest Finish Time (HEFT) [42]. OLB randomly assigns each task to the machine that is expected to be available, regardless of the estimated execution time on that machine [131]. This algorithm aims to keep all machines as busy as possible. One advantage of OLB is its simplicity. However, OLB does not consider the expected execution time of a task, the mappings it finds can result in very poor makespan. In contrast to OLB, Minimum Execution Time (MET) assigns each task to the machine with the best expected execution time, regardless of the machine's availability [129, 130]. MET aims at giving each task to its best machine. This can cause a severe load imbalance across machines. Earliest Finish Time (HEFT) algorithm aims to minimize the overall execution time of a DAG application in a heterogeneous environment. While being effective at optimizing makespan, the HEFT algorithm does not consider the budget constraint and load balancing among the machines of the cluster when making scheduling decisions. In [132], Chase et al. constructed an analytical model to quantify the network performance of workflows using cloud-based computing resources. They designed a critical-greedy algorithm to minimize the workflow end-to-end delay by defining a global budget level (GBL) parameter and preassigning tasks using the budget-level execution cost. However, this algorithm is designed for homogeneous cloud environments, where the communication time between tasks is assumed to be zero, which is not the case on heterogeneous cloud systems.

In addition to a single scheduling optimization metric, task scheduling problem becomes more challenging when two QoS metrics (i.e., time, cost and load balance) are considered simultaneously. In [133], authors have studied the problem of budget-constrained schedules of bag-of-tasks problems on clouds. They have presented a scheduler used to calculate the execution cost and performance of a workflow on multiple different clouds under budget-constraints and offer viable options to the user. They have focused on a statistical method for estimating costs. However, they have not considered the problem of load balancing. Another multi-objective algorithm named Revised Discrete Particle Swarm Optimization (RDPSO) was proposed by Wu et al. [134]. The algorithm either optimizes cost or makespan based on the budget and deadline constraints. For evaluating cost, the data transmission costs and the computation costs are taken into account. Different sets of workflows are used for experimentation, and comparisons are made with the standard PSO and the BRS(Best Resource Selection) algorithms. This multi-objective model is efficient and the results show that the RDPSO algorithm can lead to much more

cost savings and reduce the makespan than PSO and BRS algorithms.

A common drawback of the aforementioned approaches is that they do not consider the load balancing among the machines of the cluster. Our proposed E-HEFT algorithm covers all of these deficits, achieves a well-balanced load across the machines and minimizes the makespan of a given workflow under a user-specified budget constraint.

5.2.2 Heterogeneous Earliest Finish Time algorithm

Among the scheduling algorithms for heterogeneous system, the Heterogeneous Earliest Finish Time (HEFT) algorithm has been one of the most frequently cited and used because of both its simplicity and good performance [42]. HEFT is a natural extension of the classical list scheduling algorithm for homogeneous systems to cope with heterogeneity. It outperforms other comparable algorithms in terms of minimization of the execution time.

There are two phases for the algorithm. The prioritizing phase for giving a priority to each task and the machine selection phase for selecting a suitable machine that minimizes the execution time. If two or more tasks have an equal priority, then the task is selected randomly. The last phase is repeated until all tasks are scheduled on suitable machines. This algorithm has relatively low complexity and is very efficient when compared to other algorithms. Thus, HEFT assigns first a weight to each node and edge of the DAG based on the average computation and communication costs, respectively [135]. $Rank_u$ represents the length of the longest path from task t_i to the exit node, including its computational length. It is recursively defined as:

$$rank_u(t_i) = \overline{W}_i + \max_{t_j \in succ(t_i)} (\overline{c}_{i,j} + rank_u(t_j)) \tag{5.1}$$

Where $succ(t_i)$ is the set of immediate successors of the task t_i, $\overline{c}_{i,j}$ is the average communication cost of edge e_i^j, and $\overline{W}_i$ is the average execution cost for task t_i. Then, the graph is traversed upwards and $rank_u$ value is assigned to all nodes. For the exit task, $rank_u(t_{exit}) = \overline{W}_{exit}$. Tasks are then scheduled, in descending order of their rank value, on the machine which gives the smallest estimated finish time.

5.3 Mathematical formulation of the studied problem

5.3.1 System model

Recall that the problem we are interested in this chapter is performing the tasks of a workflow in a cloud computing environment while taking into account the quality of service criteria, namely the overall execution time.

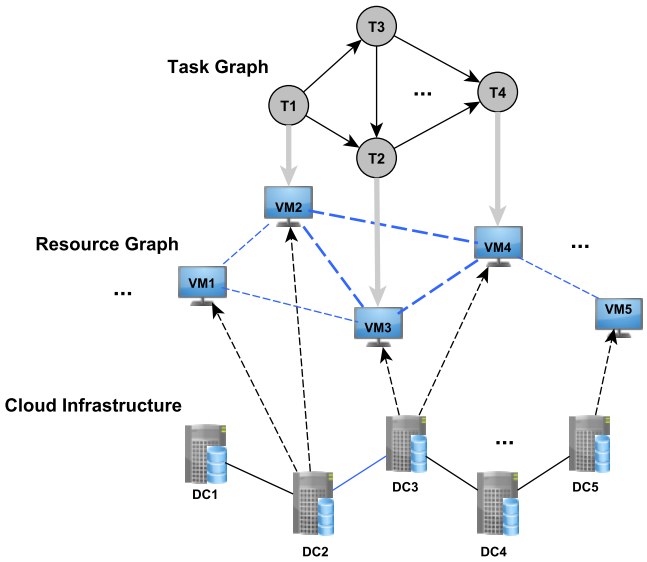


Figure 5.1 – *System model of workflow application scheduling on cloud Computing.*

As illustrated in Figure 5.1, there are three layers in the system model for the workflow scheduling problem in cloud environment [132]: (1) the task graph layer which is comprised of tasks with precedence constraints, (2) the resource graph layer which represents a network of virtual machines, (3) and the cloud infrastructure layer consisting of a set of data centers connected by network links. According to [31], the problem of task scheduling in heterogeneous systems is finding a proper assignment of tasks to machines in order to optimize some performance metrics such as resources utilization, load balancing and execution time. A popular representation of a scientific workflow application is the directed acyclic graph (DAG): $G(T, E)$, where $T = \{t_1, t_2, \cdots, t_n\}$ is the set of tasks and E is the set of directed edges between tasks. An edge e_i^j of the form (t_i, t_j) of graph G represents the precedence constraint between these tasks, case in which t_i is said to be the parent task of t_j and t_j is said to be the child task of t_i. Based on this constraint, a child task can not be executed until all of its parent tasks are fully executed. If there is

data transmission from t_i to t_j, the t_j can start only after all the data from t_i have been received. A task without parent is called an *entry* task, denoted t_{entry}, whereas a task without child is an *exit* task, denoted t_{exit}. A sample workflow is shown in Figure 5.2.

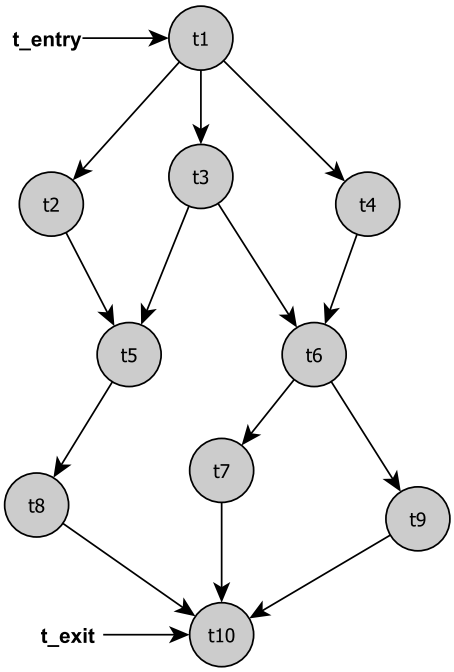


Figure 5.2 – *An example of workflow application.*

The data exchanged between the virtual machines are represented by a $n * n$ matrix which is denoted by D^{out}, where $D^{out}[i, j]$ represents the amount of transmitted data from the virtual machine $VM(t_i)$ that executes the task t_i to the virtual machine $VM(t_j)$ that executes the task t_j. We assume that the size of the output data $D^{out}_{t_i}$ produced by task t_i is known in advance. Let β be a matrix of dimension $m * m$ representing the bandwidth between the different virtual machines that execute the workflow tasks. $\beta[i, j]$ is the bandwidth between the virtual machines VM_i and VM_j. Note that this matrix is not necessarily symmetric. Indeed, the bandwidth between VM_i and VM_j is not necessarily the same as that between VM_j and VM_i. Figure 5.3 shows an example of resources graph with transfer speeds (bandwidth) between different virtual machines of a cloud environment. The transfer time between two virtual machines executing the tasks t_i and t_j, that is $VM(t_i)$ and $VM(t_j)$, is determined by the amount of transferred data and bandwidth between these virtual machines. Note that the transfer time between two tasks running

on the same VM equals 0. The transfer time TT is calculated as:

$$TT(t_i, t_j) = \frac{D^{out}[i,j]}{\beta[i,j]} \tag{5.2}$$

Let ET be a $n * m$ matrix, where $ET(t_i, VM_j)$ is the estimated execution time of the task t_i on the virtual machine VM_j. The real execution time of a task on a given virtual machine also depends on the amount of its input and output data.

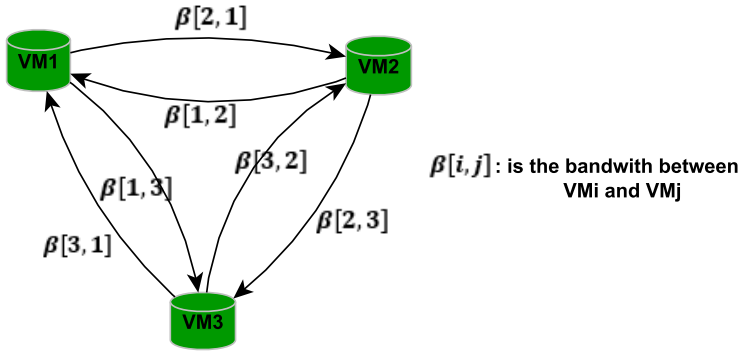


Figure 5.3 – *Architectural model.*

5.3.2 Objective function

Our objective function aims at minimizing the overall makespan of a workflow while maintaining the loand balancing. First of all we define $ST(t_i)$ and $FT(t_i)$ which are the earliest starting time and the earliest finishing time of task t_i, respectively. For the entry task t_{entry}:

$$ST(t_{entry}) = 0 \tag{5.3}$$

$$FT(t_{entry}) = ET(t_{entry}, VM_j)) \tag{5.4}$$

We recursively calculate ST and FT for the rest of tasks in the graph starting from the entry task. To calculate the earliest finishing time of task t_i, all its parents tasks (predecessors) must be scheduled taking into account the transfer time as shown in equations 5.5 and 5.6:

$$FT(t_i) = ST(t_i) + \min_{VM_i}\{ET(t_i, VM_i)\} \tag{5.5}$$

$$ST(t_i) = \max_{t_j \in pred(t_i)}\{FT(t_j) + TT(t_j, t_i)\} \tag{5.6}$$

Where $pred(t_i)$ is the set of predecessors of the task t_i.
We propose a scheduler S defined by a set of resources, a set of tasks to schedule, the total execution time, and the total execution cost, $S = (T, R, ET(t_i, VM_i), PC(t_i, VM_i))$ where:

- $T = \{t_1 \cdots t_n\}$ is the set of tasks compose the Workflow W.
- $R = \{VM_1 \cdots VM_m\}$ is the set of virtual machines.
- $ET(t_i, VM_i)$ is the total execution time of the task t_i on VM_i.
- $PC(t_i, VM_i)$:(Processing cost) is the total execution cost of the task t_i on VM_i, which equals:

$$PC(t_i, VM_i) = Cost(VM_i) * ET(t_i, VM_i) \tag{5.7}$$

Where $Cost(VM_i)$ is the cost of data processing per hour in the virtual machine VM_i, and $ET(t_i, VM_i)$ is the execution time of the task t_i on VM_i.
The objective function of the overall execution time can be defined as follows:

$$Makespan = \min\{FT(t_{exit})\} \tag{5.8}$$

Subject to:

$$\sum_{t_i \in T} PC_{t_i} \leq Budget(W) \tag{5.9}$$

$$\forall t \in T \mid \exists!\ VM_i \in R\ ,\ \mu(t_i) = VM_i \tag{5.10}$$

$$\sum_{t_i \in r_j} RAM(t_i) \leq RAM(VM_i) \tag{5.11}$$

$$\sum_{t_i \in r_j} IS(t_i) + OS(t_i) \leq SC(VM_i) \tag{5.12}$$

The associated constraints are given in Eq. 5.9 to 5.12. Where $IS(t_i)$ is the input size of a task t_i, $OS(t_i)$ is the output size of a task t_i and $SC(VM_i)$ is the storage capacity of VM_i.

Constraint 5.9 ensures that the workflow processing cost must be less or equal to the budget dedicated to this workflow.

Constraint 5.10 ensures that each task is scheduled on a single virtual machine. Where μ is a matching function between the set of tasks and the set of virtual machines, which indicates for each task the machine on which it will be launched.

Constraint 5.11 ensures that the total RAM required by all tasks running on VM_i should be less than the available RAM on this virtual machine.

Constraint 5.12 ensures that the total disk space (input size and output size) required by all tasks assigned to VM_i should be less than the storage capacity available on VM_i.

5.4 E-HEFT: Enhancement Heterogeneous Earliest Finish Time algorithm

The algorithm is consisted of four phases. In the first phase, we specify the load threshold of each machine based on both processing speed and storage capacity. In the second phase, we define the datasets dependencies in order to cluster the datasets into different datacenters based on these dependencies. In the third phase, we group the set of tasks by level on the graph, then we assign a value (Rank) for each task based on the *Rank* function of the HEFT algorithm. Finally, we schedule these tasks on its best machine based on *Matching Game* theory.

5.4.1 Attribution of VMs threshold phase

The threshold represents the utilization storage percentage for each machine that should not be exceeded. We will attribute a threshold for each machine on the cluster based on its storage capacity and data processing speed. For quantifying and evaluating the performance of each machine, we set up the HPC Challenge (HPCC). It is a benchmark designed to give a picture of overall computer performance including floating point computer power, memory subsystem performance and global network issues. It consists of seven separate workloads building on the success of the TOP500 list Linpack HPL based workload [1].

In this chapter, we first use the HPL (High-Performance Linpack Benchmark) [77] measurement as a performance value to know the real datacenter performance on the HPCC benchmark. Then, we record the execution time of each machine based on the output of HPL benchmark. After that, the shortest execution time is used as a reference to normalize the execution time measurements, in such a way we attribute randomly the highest threshold to the machine that has the shortest execution time. Then, the

[1] https://www.top500.org/lists/

normalized values are used to distribute an appropriate threshold to each machine on the cluster. Thus that a high-speed machine will handle tasks more than low-speed machine.

5.4.2 Datasets dependencies specification phase

In our strategy, we initially adapt a dependency matrix to represent the affinity between the datasets. Cloud workflows can be complex, the execution of one task might require many datasets. Furthermore, one dataset might also be required by many tasks. So, we say that the datasets d_i and d_j have a dependency if there are tasks that will use them together. The dependency degree is the total number of tasks that use both d_i and d_j. We use $dependency_{ij}$ to denote the dependency between the datasets d_i and d_j and the quantity of this dependency is the number of tasks that use both d_i and d_j . It can be calculated by counting the tasks in common between the task sets of d_i and d_j, which are denoted as T_i and T_j.

$$dependency_{ij} = card(T_i \cap T_j) \tag{5.13}$$

The dependency matrix is defined by: $DM = \{dependency_{i,j}, \quad (1 \leqslant i, j \leqslant n)\}$. For the elements in the diagonal of DM, each value means the number of tasks that will use this dataset. DM is a symmetric matrix of dimension $m * m$, where m is the total number of existing datasets. We store the datasets that have the highest dependency degree in the same datacenter based on the total number of tasks that use these datasets as shown in formula 5.14.

$$Max(dependency_{ij}) \tag{5.14}$$

As an example in Figure 5.4 (left side), we assume that we have four datasets (d_1, d_2, d_3 and d_4), and we have also five tasks to be processed (t_1 to t_5), the dependency matrix DM is shown in Figure 5.4 (right side).

5.4.3 Tasks sorting phase

The objective of this phase is to define the different levels of a given workflow (DAG). Therefore, tasks belonging to the same level can be execute concurrently. This is because two tasks at the same level do not exchange data (or they are not linked by precedence constraints). Then, a weight is assigned to each node and edge of the graph, depending on processing length of each task in all cluster's machines and communication length

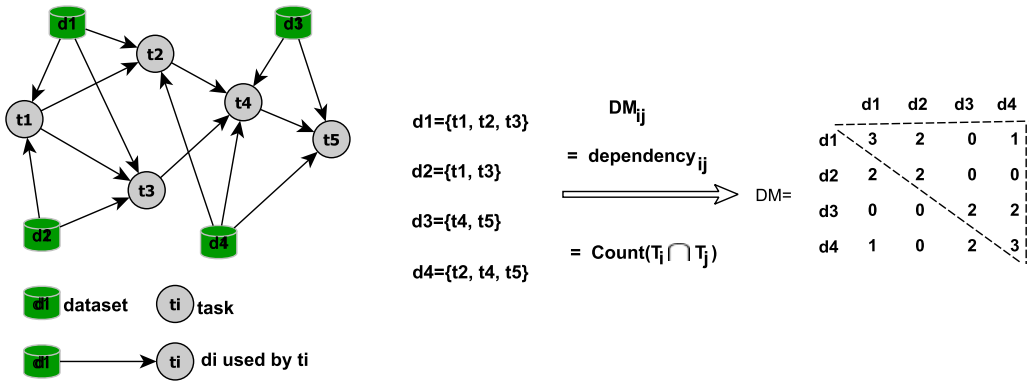


Figure 5.4 – *An example of dependecy matrix.*

between tasks. Then, the graph is traversed upwards and a value (rank) is assigned to each node at each level. The first level and the last level contain respectively the exit task t_{exit} and the entry task t_{entry}. Tasks are sorting in descending order in the list based on their rank value.

For calculating rank value to give priority for each task to be executed, we use the rank function of the HEFT algorithm. **Algorithm 6** shows the pseudo code of the ranking method of HEFT algorithm.

Algorithm 6: HEFT ranking algorithm

for *each task* t_i *in the graph (DAG)* **do**
  calculate average execution time on all VMs
  if *task* t_i *is the last task* **then**
    rank value of t_i = its average execution time
  else
    $rank_u(t_i) = \overline{W}_i + \max_{t_j \in succ(t_i)} (\overline{c}_{i,j} + rank_u(t_j))$

5.4.4 Virtual machine selection phase

The final phase of the proposed approach is to choose an "optimal" virtual machine to run each of the workflow tasks. We use game theory [97], which is a theoretical approach of the game that provides an appropriate solid mathematical tools to study the considered issue and to allocate resources to the tasks on the basis of low cost, load balancing and improved tasks execution time. So far, we have modeled the problem of virtual machines selection as a many-to-one matching game where the players are the tasks and the

VMs. Each task has the right to choose one VM. By contrast, the VMs can host one or more task respecting the maximum number of quota q_{vm}.

This choice based on the preferences list for all tasks and virtual machines. The preference lists of each task on the workflow $PL(t_i) = \{VM_{j*}, \cdots, \}$ contains the set of VMs, which are selected like potential sites. The elements in preference list $PL(t_i)$ are put in an ascending order according to the processing power of the machines, their cost processor, memory and storage capacities. So that the machine having the best values of these parameters depending on the requirement of the client has the highest rank. On the other hand, The preference list of the j^{th} virtual machine VM_j is $PL(VM_j) = \{t_{i*}, \cdots, \}$. These tasks are sorted in ascending orders according to different parameters as we have seen in chapter 4. The tasks are initially prioritized according to the impact of these tasks on the use of the virtual machine in terms of CPU, RAM, etc, such that one having least impact has highest rank. Thus, the key factor for prioritizing tasks is their CPU utilization ratio. As a solution of virtual machines selection phase we used the matching algorithm as shown in **algorithm** 7 based on the deferred acceptance algorithm introduced in [**?**], which is a well-known approach to solving the standard matching games.

Algorithm 7: Task-VM matching algorithm

Input: Tasks' preference lists, VMs' preference lists .

1 **while** *(∃ a free task t_i that still has a VM_j to propose to)* **do**
2 Task t_i proposes to all VMs on its preference list.
3 **if** *($q_{vm_j} \geq 0$)* **then**
4 t_i is assigned to the waiting list of VM_j
5 **else**
6 Compare t_i with the current tasks on the waiting list for the VM_j and reject the task that has least preference.
7 The rejected tasks re-apply to their next best choice.
8 Each VM update its waiting list based on its preference list, and rejects the rest.

Output: Every t_i is on a waiting list of one of the VMs, and,thus, we have convergence to a stable matching.

An overview of our approach is given by **algorithm** 8 which includes all the aforementioned phases.

Algorithm 8: Task scheduling algorithm

Input: Workflow W, set of virtual machines VMs
Attribute a threshold for each virtual machine.
Defining datasets dependencies by using the dependency matrix.
Read the DAG
Break W into set of levels L by traversing the DAG upward.
$K \leftarrow 1$; // first level
while $(K \leq L)$ **do**
    for *each task in level k* **do**
        Apply HEFT ranking algorithm (algorithm 6)
    Sort the tasks in a scheduling list by descending order of $rank_u$ values
    Check tasks and virttual machines Preferences Lists(PL)
    Assign task t_i to the best virtual machine VM based on preferences list.
    Apply Task-VM matching algorithm (algorithm 7)
    Remove t_i from the level K
    $K \leftarrow K + 1$
return Set of VMs with the mapping tasks.

5.5 Experiment result and analysis

In this section, we present the experiments conducted in order to evaluate the performance of the proposed E-HEFT algorithm. Performance metrics, experimental setup and results are shown in the following subsections.

5.5.1 Experimental setting

To evaluate a workflow scheduling algorithm, we should measure its performance on some sample workflows. So, there is a need for a good simulator for experimental purposes. One such a simulator is CloudSim [127], which has been widely adopted for the modeling and the evaluation of cloud-based solutions. Particularly, CloudSim provides a generic broker modeled as a class named DataCenterBroker, we extended this class to support DAG-structured workflows and to model the behavior of this component and its particular placement policies. On the other hands, CloudSim is an extensible simulation toolkit that enables modeling and simulation of cloud computing systems and application provisioning environments. The Cloudsim could implement generic application provisioning techniques that can be extended easily with limited efforts. According to the implementation using Cloudsim, the VMs are considered as the cloud resources

and Cloudlets as tasks/jobs. We ran our experiment on one datacenter that contains 20 VMs, and the configuration of virtual machine is shown in Table 5.1. All experiments are performed on a Pentium(R) Dual-Core processor with a speed of 2.8GHz and memory of 16GB.

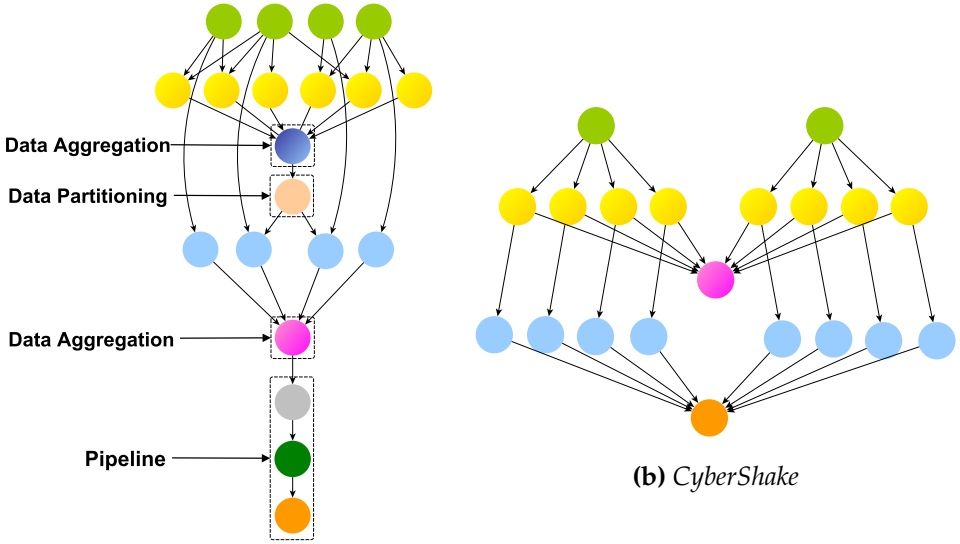
Table 5.1 – *Initial parameters of virtual machine*

parameters of virtual machine	**Value**
Number of VM cores	4
Mips of each core	2000MI
RAM	1GB

In order to make the results more realistic, it is important to conduct experiments using workload traces from a real system. Bharathi et al. [136] investigate the characterizations of six realistic workflows from diverse scientific applications, two of which are used in our experiments, which are Montage for astronomy and CyberShake for earthquake science. Montage [33] is an astronomy application that was created by the NASA/IPAC Infrared Science Archive as an open source toolkit that can be used to construct large image mosaics of the sky using input images in the Flexible Image Transport System (FITS) format. The CyberShake [34] workflow is a seismology application that calculates Probabilistic Seismic Hazard curves for geographic sites in the Southern California region. Four different sizes of these workflows are chosen, small (around 30 tasks), medium (around 100 tasks), large (1000 tasks) and x-large (10000 tasks). Figure 5.5 shows the approximate structure of a small instance of each workflow used in our experiments. For each workflow, tasks with the same color belong to the same type. It can be seen that these two workflows have different structures, data and computational requirements.

5.5.2 Competitive algorithms and performance metrics

We compared our E-HEFT algorithm with the Heterogeneous Earliest Finish Time (HEFT) [42] and MinMin Task Scheduling Heuristic (MinMin-TSH) [137]. The MinMin-TSH algorithm is a widely-used scheduling algorithm. This is a task-based greedy algorithm that allocates each ready-to-run task to a machine based only on the local information of that task. There are two phases in MinMin-TSH algorithm. In the first phase, it finds the resource with minimum execution time of all tasks. Then, in the second phase, it selects the task with minimum execution time among all the tasks and schedules it on that resource. The same procedure is repeated by MinMin-TSH algorithm until all tasks



(a) *Montage*

(b) *CyberShake*

Figure 5.5 – *The structure of two realistic scientific workflows.*

are scheduled. We have chosen MinMin-TSH [137] and HEFT [42] algorithms because they produce efficient schedules and have been used as baselines in many related works.

Our scheduling algorithm aims at minimizing the total execution time of the tasks as well as achieving a well-balanced load across all VMs. Hence, we have used two metrics to evaluate our scheduling algorithm, which are the total execution time (makespan) and the degree of imbalance (DI).

- **Makespan**:is the maximum needed time to complete the execution of all tasks.

$$makespan = \max_{i \in \{1,\cdots k\}} \{FT(t^i_{exit})\} \tag{5.15}$$

  Where t^i_{exit} , $i \in \{1, \cdots k\}$ are tasks that don't have any precedence constraints, so they can run in parallel to complete the execution of the entire workflow.

- **Degree of imbalance (DI):** to measure degree of imbalance (DI) of all virtual machines on the cluster, we use the following formula:

$$DI = \frac{L_{max} - L_{min}}{L_{avg}} \tag{5.16}$$

Where L is a load of a VM on the cluster, and equals:

$$L = \frac{tasklength(VM_j)}{C(VM_j)} \tag{5.17}$$

Where $tasklength(VM_j)$ is the total length of tasks submitted to the VM_j and $C(VM_j) = pe_{numj} * pe_{mipsj}$ is the capacity of a VM_j, with pe_{numj} is the number of processors in VM_j and pe_{mipsj} is the million instructions per second of all processors in VM_j.

5.5.3 Experimental results with simulaion

The makespan results (the average of 5 executions) for the Montage and Cybershake workflows are shown in Figures 5.6 and 5.7, while Figure 5.8 shows the comparison of degree of imbalance between E-HEFT, HEFT and MinMin-TSH algorithms.

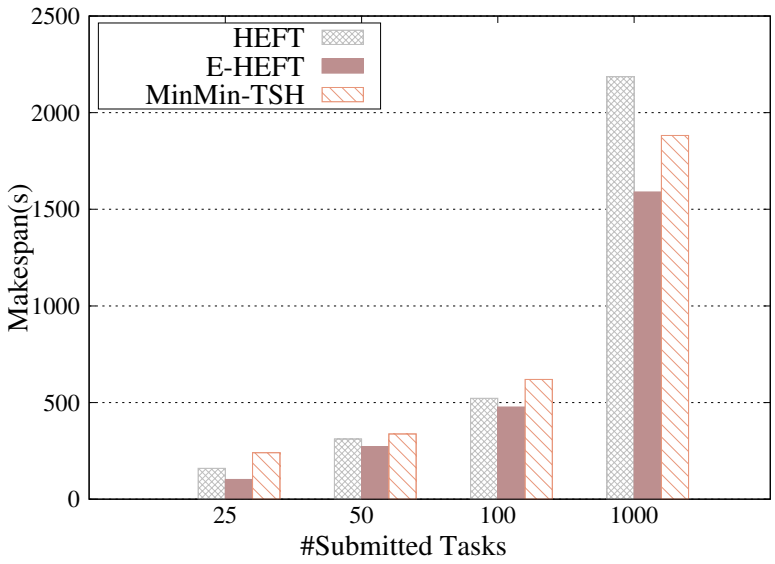


Figure 5.6 – *Average makespan of the three algorithms based on Montage workflow.*

According to the experimental results in Figures 5.6 and 5.7, it can be seen that the total running time of scientific workflows lengthens when the number of submitted tasks increases. It is clearly evident from the graphs plotted in Figures 5.6 and 5.7 that E-HEFT is more efficient when compared with other two algorithms based on both workflows. According to Montage workflow, our proposed algorithm (E-HEFT) outperforms HEFT and MinMin-TSH algorithms in terms of the average makespan of the submitted tasks by 21.37%, and 28.98%, respectively as shown in Figure 5.6. On the other hand, The E-HEFT showed an average improvement of 26.07% compared to HEFT and 21.93% com-

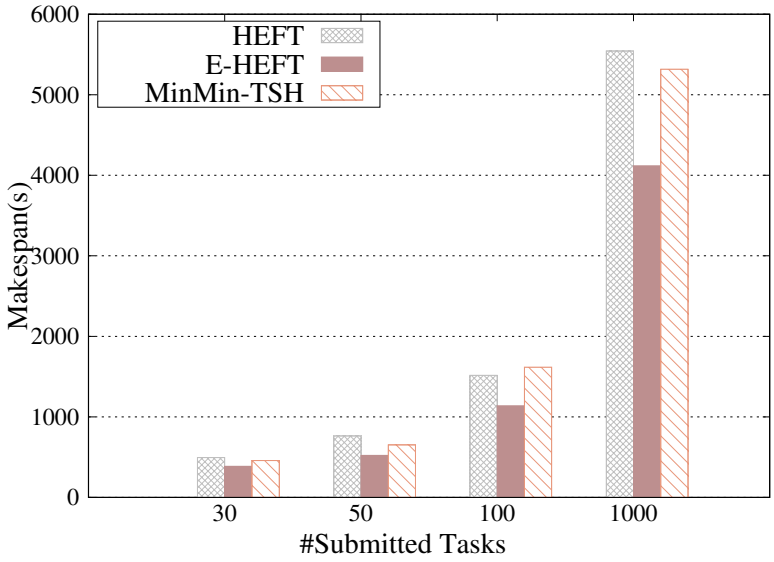


Figure 5.7 – *Average makespan of the three algorithms based on CyberShake workflow.*

pared to MinMin-TSH based on Cybershake workflow as shown in Figure 5.7. It is worth mentioning that makespan is closely pertinent to the volume of data movement for data intensive workflow. So, we can state that those improvements variations are mainly due to the volume of data that is transferred among activities in each workflow. Compared with two other algorithms, our algorithm incurs less data movement during the workflow execution thanks to the datasets dependencies specified in the second phase of our algorithm.

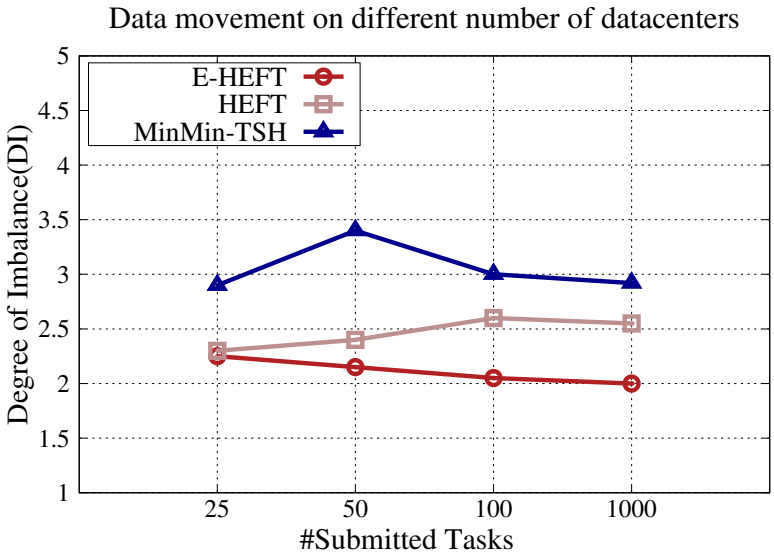


Figure 5.8 – *Comparison between algorithms (E-HEFT, HEFT and MinMin-TSH) based on degree of imbalance.*

From Figure 5.8, it can be seen that our algorithm (E-HEFT) can effectively balance the load among all virtual machines and has a lesser degree of imbalance when compared with two other algorithms. This is due to the attribution of load threshold to the VMs as specified in the first phase of our algorithm.

5.6 Conclusion

In this chapter, we have proposed an enhancement of the Heterogeneous Earliest Finish Time algorithm (E-HEFT) for achieving tasks scheduling with load balancing among virtual machines and minimum execution time. The algorithm attributes a utilization threshold to each VM that it should not exceed. Then, it stores the dependent datasets on the same datacenter. Next, tasks are sorted based on their rank value using HEFT algorithm. Finally, matching game theory is used to select the suitable virtual machines to execute the submitted tasks. To evaluate the performance of our algorithm, we compared it with some existing techniques based on the execution time (i.e., makespan) and degree of imbalance (DI). Experimental results show that our algorithm not only balances the load, but also takes less time to execute tasks compared to the two other algorithms.

Efficient task scheduling is essential for achieving good system performance mainly in distributed systems such as Hadoop MapReduce. Hadoop MapReduce is considered as one of the most well-used distributed platforms for processing large-scale data in cloud environment. The scheduler in Hadoop manages and monitors the scheduling of tasks. In addition, if a failure takes place, Hadoop reschedules the failed tasks. This makes fault tolerance a critical issue for the efficient execution of any application running on Hadoop in order to ensure the quality of service (QoS) and to meet the end-users expectations. In the next chapter, we intend to lead a better understanding of such fault tolerance mechanism despite failures. Towards this direction, we will analyze the performance of many representative Hadoop MapReduce applications, with different execution parameters under different failure scenarios. We will also present different options to inject failures into MapReduce applications in order to simulate real world failures.

Chapter 6

Hadoop Scheduling Under Different Types of Failure

Contents

In this chapter, we aim at understanding the fault tolerance mechanism of Hadoop MapReduce with different execution parameters and under different failure scenarios. Next, we will analyze the impact of checkpointing interval selection, which is a fault tolerance technique, on the performance of Hadoop. Hence, two different scenarios have been simulated. The first consists in analyzing the performance of Hadoop in the presence of different types of failure (task failure, TaskTracker failure and NameNode failure). The second consists in investigating the impact of checkpointing interval selection on the performance of Hadoop under various failure probabilities. This chapter is derived from:

– **Yassir Samadi**, Mostapha Zbakh and Claude Tadonki. "Performance comparison between Hadoop and Spark frameworks using HiBench benchmarks". Concurrency and Computation: Practice and Experience. 2017; e4367.
https://doi.org/10.1002/cpe.4367.

– **Yassir Samadi** and Mostapha Zbakh. "The impact of checkpointing interval selection on the scheduling performance of Hadoop framework". To appear in ICMCS 2018: The 6th International Conference on Multimedia Computing and Systems. IEEE, Rabat, Morocco.

6.1 Introduction

Nowadays, the technology of cloud computing has been extensively used by many enterprises such as Facebook and Google, to name a few. By using the MapReduce framework, these enterprises can process enormous volumes of data on a large clusters. MapReduce is the most popular distributed paradigm thanks to its features such as fault tolerance for processing of large-scale data [138, 139], Hadoop is considred as a widely used implementation of MapReduce, it provides an open-source solution for processing with big data [1]. Hadoop framework contains two important components which are MapReduce and Hadoop Distributed File System (HDFS) [140, 141, 142]. The main payoff of the distributed computing platforms like Hadoop MapReduce is that they hide complex configurations in libraries and let the developers focusing only on their applications. Hadoop is one of the most well-known distributed platforms for processing large-scale data, which attracts researchers' attention to improve its existing features such as fault tolerance and to develop new functionalities as well [143]. Due to the growing volume of data at a dynamical rate by nature, the system will experience more failures. Authors in [144] claim that running a complex tasks on a Hadoop cluster may trigger many software failures. In addition, several reasons my cause a task failures on the scheduler of Hadoop such as bugs, missing data, etc.

Fault tolerance [145] is the case when a system is capable of providing the expected services despite the presence of faults that may cause errors in the system itself. Fault tolerance includes methods and techniques to ensure the availability of resources on distributed systems. In other words, by applying the fault tolerance mechanism, we can get

[1] http://Hadoop.apache.org

back the damaged data or the faild tasks to the last normal state before the accident. Fault tolerance on distributed systems is essential to ensure the performance of tasks processing. When considering the nature of distributed systems such as cloud computing which contains hundreds (and even thousands) of servers, virtual machines, routers, etc. failures can occur with a high probability. In case of its absence, the system might face failures which can deteriorate the system performance and violate QoS requirements. Therefore, an efficient fault tolerance techniques are required to prevent failures.

Several studies have been dedicated to improve the performance of distributed systems under failures. Authors in [146], have claimed that the main reason of failures is the lack of information sharing about failures between nodes (e.g., bugs, TaskTracker failure, etc.) after analyzing the performance of Hadoop under failure. Then, they have presented a RCMP strategy that performs efficient task re-computation in case of failures for big data analytics. It just re-executes the failed tasks instead of replicates data. However, RCMP only focuses on Input/Output heavy jobs, so we cannot apply this strategy on all kinds of Hadoop jobs. Jin et al. [147] proposed a stochastic performance model to study how node failures influence MapReduce applications. They considered the best-case scenario and the worst-case scenario under the assumption that the inter-arrival time of node failures follows exponential distribution to estimate the expected execution time of tasks. The simulation results show that MapReduce job with three replicas can achieve better performance than that with one replica, because when there is more data replicas in different nodes, the amount of data migration will be reduced when rescheduling jobs at failure time. In [148], authors presented a fault tolerance technique called (BeTL) adopted on the top of Hadoop framework. They proposed to send the meta-data to master as checkpoint and they come up with a solution to deal with the instability of the generation spills. Experiment results show that BeTL technique gives results better than Hadoop with 6% in case of no failure detected and 4% to 51% in the presence of failures. However, a node failure cannot be well handled since the intermediate data are not replicated at runtime.

Although Hadoop has adopted an efficient fault tolerance mechanism for dealing with many types of failures, it still experiences a lot of failures because of the nature of the cloud computing. Generally in Hadoop, the detection of node failures takes place after an interval of 10 min, which is the default interval for Hadoop to declare a node is dead. Then, the failed tasks wil be re-executed on other available nodes. The re-execution of failed tasks may cause an increase of the job execution time which can affect negatively

the performance of the scheduler on Hadoop. Generally speaking, there are two kinds of fault tolerant techniques used in Hadoop framework which are replication and checkpointing. As for replication, each dataset will be replicated on the cluster and used as a backup, 3 replicas by default. So, a secondary node is necessary. Either the two nodes run simultaneously or the secondary node keeps on stand by until the primary node fails over, the resource should be doubled to meet the requirement of fault tolerance. Another solution is checkpointing [149], which is an efficient fault tolerant approach as it enables the failed tasks to be resumed from just a previous consistent state and not from scratch. During the checkpointing, an application is responsible for writing its current state periodically at regular intervals to a local disk during the runtime stage, so in case if this application failed to run, it resumed from the last valid state rather than from the scratch.

Although the checkpointing is considered as one of the most efficient fault tolerance technique, unnecessary frequent checkpoint might deteriorate the system performance. Consequently, the checkpointing interval, i.e., the amount of time between two consecutive chekpoints, must be selected taking into account the failure probability, as well as the nature of the workload. Towards this direction, this chapter intends to provide a better understanding of Hadoop fault tolerance mechanism by quantifying the impact of failures on MapReduce applications. In particular, we analyze the performance of Hadoop with presence of different types of failures (task failure, TaskTracker failure and NameNode failure) [150]. Next, we discuss various approaches to inject failures into a MapReduce jobs and evaluate the performance penalty depending on the type of failure injection, the frequency of failure, the function being affected by the failure, and various system and application parameters. We then investigate via simulation the impact of checkpointing interval selection on the performance of Hadoop under various failure probabilities in order to demonstrate that choosing the optimal checkpointing interval has an influence on task performance.

This chapter begins by presenting background of Hadoop MapReduce in section 6.2. Section 6.3 briefly discusses the fault tolerance mechanism of Hadoop MapReduce as well as the checkpointing technique. In section 6.4, simulations are conducted to analyze the fault tolerance mechanism of Hadoop Mapreduce under different type of failures as well as the impact of checkpointing interval selection on the performance of Hadoop. Section 6.5 presents the conclusion and our future work.

6.2 Overview of Hadoop MapReduce

6.2.1 Definition

MapReduce [138, 151] is a programming paradigm for distributed computing based on Java, introduced by Google to process large volumes of data. The model is relatively simple. To maximize potential parallelism, it consists of two important stages. The first one (Map) is the name of a high-level function. It applies a given function to each item of a list and returns a list of pairs in the form of (key,value). The result of this phase is stored on a local disk and it does not stored on HDFS (Hadoop Distributed File System). The second stage (Reduce) is the name of a high-level function that applies a given function to all elements of a list and returns a single list. Input data is split into "small" units (64 MB by default) [152], each being treated in parallel by a Map function. The result of the treatment is sorted by key to form data units for the Reduce function. The output of map phase is grouped by key and transferred from the mappers to the reducers (shuffling phase). Next, the reducer takes the input as $(K, List(v))$ and generates the output as (K, V). Then, the output data is written to HDFS. It is possible to distribute data processing because MapReduce programs are inherently designed to be executed in parallel. Figure 6.1 depicts the phases that MapReduce follows to execute the jobs. MapReduce framework operates on $< key, value >$ pairs as shown bellow:

$$map(key1, val1) \leftarrow list(key2, val2)$$

$$reduce(key2, list(val2)) \leftarrow list(key3, val3)$$

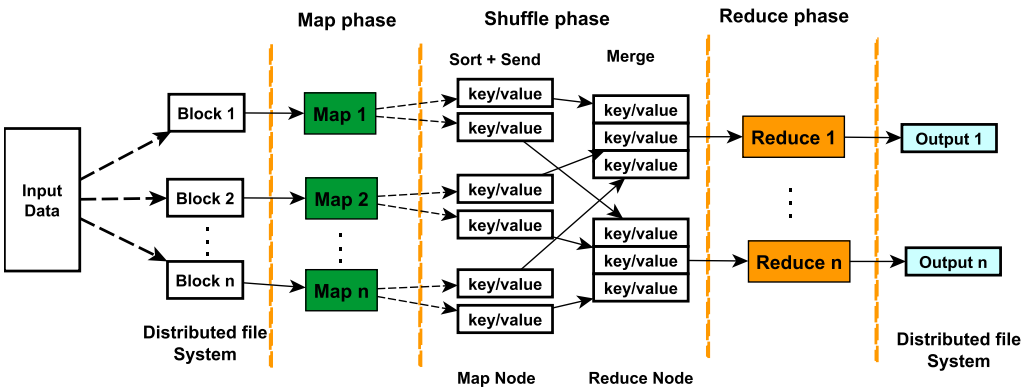


Figure 6.1 – *MapReduce Programming Model*

In this chapter, we focus on the open-source MapReduce implementation, Hadoop [2]. Hadoop uses master/slave scheduling mode. In this architecture, there is a master node which is the JobTracker and several worker nodes which are TaskTrackers. JobTracker receives requests from MapReduce clients and assigns them to the available TaskTrackers for execution. TaskTracker, in his trun, tracks the execution of MapReduce tasks running locally and sends statuses update to the JobTracker.

Hadoop consists of the following main components: (1) JobTracker, which is responsible for assigning tasks, (2) TaskTracker, which is responsible for tasks execution, (3) NameNode, which is a master server that manages the file system namespace and regulates access to files by the clients, and (4) DataNode, which manages the storage attached to the hosting nodes. The JobTracker and NameNode daemons run on the Hadoop master node, while the TaskTracker and DataNode daemons run on the slave nodes. Figure 6.2 shows a simplified overview of Hadoop.

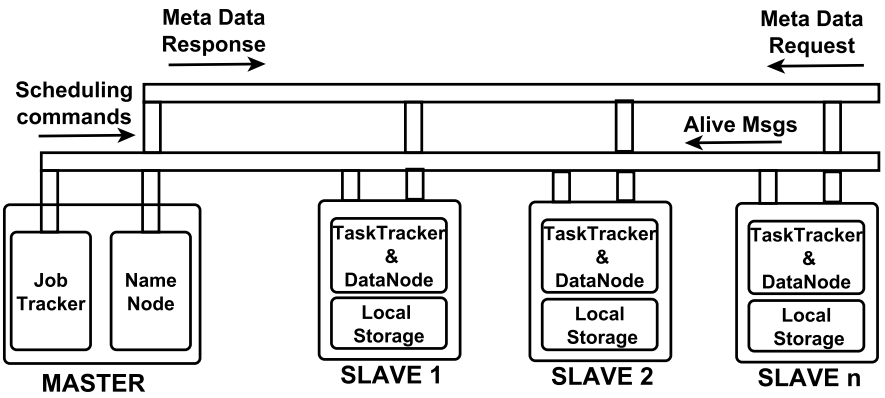


Figure 6.2 – *Intercations between Hadoop components*

6.2.2 Type of failures in Hadoop MapReduce

The occurrence of failures in distributed systems such as Hadoop may happen due to several reasons [153]. Google has announced that there are several nodes that fail every day in Google cluster and Hadoop is bit familiar with the Google file system (GFS). Hadoop distributed file system (HDFS) tends to be highly fault tolerant. Generally speaking, there are three types of failure in Hadoop MapReduce: task failure, TaskTrackers (workers node) failure and JobTracker (master node) failure:

[2]http://Hadoop.apache.org

- **Task Failures:** In Hadoop, a master node tracks the status of each task running on the cluster by creating associated data structures. The task failure on Hadoop is caused by the component's failure on which the task is belonging to. A task failure can happen when a map or a reduce phase is interrupted. There are several reasons to have a task failure. Among these reasons, there are bad records, contention, media corruption, and bugs. For instance, in Figure 6.3, the map task of Job3 exceeded its maximum number of scheduling attempts which was the cause of the fail of all reduce tasks. As a consequence of this map task failure, the entire job3 failed.

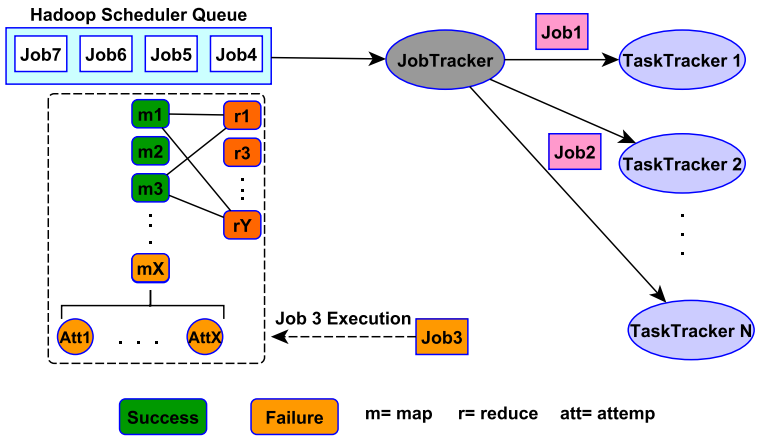


Figure 6.3 – *Example of Hadoop MapReduce task failure*

- **TaskTracker failures:** A TaskTracker failure occurs when it stops accepting tasks from the JobTracker. There are several reasons to have a TaskTracker failure such as hard disk failure, memory errors, etc.
- **JobTracker Failures:** The failure of JobTracker is the most serious failure mode because the JobTracker is a single point of failure in Hadoop [154]. So, in this case the entire job fails. Moreover, the state of the current task is lost.

Table 6.1 summarized the aforementioned type of failures.

6.3 Fault tolerance techniques

In this section, we define the fault tolerance mechanism. We present the fault tolerance technique adopted by Hadoop to deal with failures as well as the fault tolerance technique based on checkpointing.

Source of failure	Cause of failure	Loss
Task	bad records, media corruption, and bugs	Task
TaskTracker	hardware failures, e.g. memory errors, hard disk failures, or overheating CPUs	Task set
JobTracker	Crashing node	Entire job

Table 6.1 – *Type of failures in Hadoop MapReduce*

6.3.1 Definition

Fault tolerance allows a system to work properly without losing data even if some components failed [155]. In other words, fault tolerance is a strategy used for detecting and identifying faults when they occur on a system, then it aims at recovering failed tasks without any damage or data lost. There exists a lot of unpredicted events that can be appeared at runtime. Hence, achieving 100% of fault tolerance is a hard task. However, the main objective of fault tolerance techniques is to avoid frequent failures. Without loss of generality, in distrbuted systems, there exists two kinds of fault tolerance techniques, which are:

- **Reactive fault tolerance:** This policy reduces the effect of failures during the task execution by using checkpointing to tolerate the fault effectively when a failure occurs.
- **Proactive fault tolerance:** This policy aims at predicting faults before they actually happen by analyzing resources such as tasks and VMs. Therefore, this policy allow to avoid faults in advance.

6.3.2 Fault tolerance in Hadoop MapReduce

Node and task failures are prone to happen during a MapReduce job execution process. When a task failure occurs, slave node (worker) sends a heartbeat message to inform the master of the task failure. To efficiently deal with failures, the master node, in his turn, will re-execute the failed tasks on any healthy node as soon as possible. The task re-execution can increase the job execution time and waste resources. As an example, a task with a makespan of 120s, its makespan can go to 600s if a TaskTracker failure occurs, and 900s if a DataNode fails [146]. Moreover, as we have said before, the main reason of

failures is that nodes do not share information about failures between each other, which can lead to fail all tasks of the job [156].

In the case if TaskTracker occurs, the master (JobTracker) relies on periodical communication with all workers to detect worker (TaskTracker) failures. Authors in [145] have demonstrated that a JobTracker detects the failure of TaskTrackers with delay due to the communication nature between JobTracker and TaskTrackers. Theoretically, TaskTrackers send heartbeat messages to JobTracker (NameNode) every 3 second. The JobTracker in his turn checks the availability of TaskTrackers every 200 second. If it passed about 10 minutes without any response from the TaskTracker, so it is declared dead. Next, the JobTracker re-executes the failed tasks (running on the dead TaskTracker) on another TaskTrackers according to their availability.

When a JobTracker fails, MapReduce will restart the master node. After restarting a JobTracker, all jobs that were running on this node are stopped and need to be resubmitted. This kind of rescheduling method is simple but often increases the task execution time and the processing cost. In Figure 6.4, we show a big picture of the default fault tolerance mechanism on MapReduce.

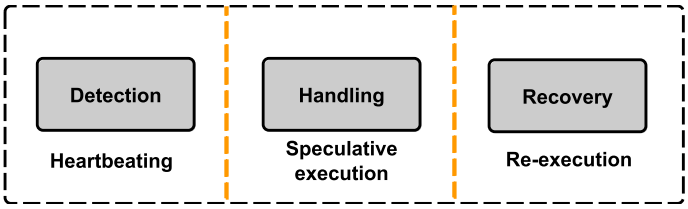


Figure 6.4 – *Fault Tolerance Mechanism in Hadoop.*

6.3.3 Fault tolerance based on checkpointing technique

In order to deal with failures, provide a good service performance, and meet the quality of services (QoS) requirements, we should design an effective and efficient fault tolerance mechanism. In most current cloud platforms, checkpointing [157], which periodically saves the state of an application to a local storage, and replication, which replicates data on different node of the cluster, are considered the most used fault tolerance mechanisms. In this chapter, we only discuss the fault tolerance based on checkpointing technique. Checkpoint is an operation that stores the current state of computation in a stable storage. In distributed systems such as Hadoop MapReduce, checkpointing technique allows working nodes to prevent the progress of failures. In other words, if a failure oc-

curs, the system will resume the failed tasks from the last chekpointing which reduces the overhead of computations.

Each set of parallel local checkpoints incurs a checkpointing overhead (CP), which is considered to be a linear function of the job's parallel service time. We get the checkpointing overhead by counting the total number of checkpoints on the longest runtime stage without failures. Specifically, it is defined as:

$$CP = CPF * S \tag{6.1}$$

Where CPF is the factor checkpointing overhead of a job whereas S is the parallel service job's time. ΔT represents the checkpointing interval during the runtime stage af an application, which can have a value in $]0, 1[$. So, we compute the number of checkpoints NCP that each job performs at runtime based on the checkpointing interval:

$$NCP = \left(\frac{1}{\Delta T}\right) - 1 \tag{6.2}$$

As a sample example, we consider that the checkpointing interval $\Delta T = 0.25$ and the parallel service time of a job is S=1, the number of checkpoints will be NCP = 4-1=3 and a checkpoint will be created after the 30%, 60% and 90% of the job's service time is completed. In this case, the parallel service time S of a job indicates only its computational work and does not include checkpointing overhead. Consequently, we assume in other case that the checkpointing overhead factor equals CPF = 0.05, so the total execution time of the same job will equal $T = S + NCP * CPF$, that is $T = 1.15$.

Fault tolerance in distributed systems such as Hadoop MapReduce is usually achieved via application directed checkpointing, which is more practical, system-independent and easier to implement. For instance, there are two imprtant components in MapReduce, which are: (1) the JobTracker, which is responsible for managing jobs, and (2) TaskTrackers, which are slave nodes responsible for executing tasks assigned from the JobTracker as we have seen in section 6.2.1. The master and worker nodes communicate with each other by exchanging a heartbeat messages. The master node sends periodically a heartbeat message for each worker nodes (TaskTrackers) to check whether it still alive or note during the runtime stage. At the same time, each TaskTracker sends its current status to the master node (JobTracker). In case of failure, the TobTracker then will assign failed map/reduce tasks to an idle TaskTracker depending on its capacity. In case if the Job-

Tracker do not receive a response from a TaskTracker during the heartbeat interval due to a failure, it will declare this node as failed. To guarantee the availability and reliability of resources on the cluster, the master node (JobTracker) will re-execute the failed tasks running on the failed nodes on another available worker nodes.

Checkpointing is the most widely adopted fault tolerance technique in large-scale distributed systems, such as Hadoop MapReduce [149] because it ultimately yields good performance. However, we should efficiently choose the checkpointing interval (i.e. the time between two consecutive checkpoints). In order to run efficiently an application, the checkpointing interval should be assigned taking into account the application nature and its parameters, the failure probability as well as the system features on which this application is running. Infrequent checkpointing may lead to greater recovery time and thus poorer performance. So, selecting an appropriate checkpointing interval is not a trivial task. Towards this direction, we conduct in this chapter a simulation study to show how the checkpointing interval selection can affect the system performance.

6.4 Experiment setup and result analysis

6.4.1 Experimental setup

For our experiments, we deployed a Hadoop cluster using 5 nodes. Each node is a virtual machine with 3 core and 2 gigabytes memory, see Table 6.2 for reference. We run Hadoop 2.7.4 with the default configuration in Centos Linux. One node in the cluster acts as a master node and the rest act as slave nodes (workers). The master node is reserved for JobTracker and NameNode, the rest of nodes run DataNode and TaskTracker processes. TaskTrackers were configured with 8 slots for running map tasks and 4 slots for reduce tasks. At the level of Hadoop Distributed File System (HDFS), a chunk size of 128 MB is used. We set a replication factor of 2 for input/output data.

6.4.2 Failure injection tool

To evaluate the fault-tolerance in Hadoop MapReduce, several things should be specified such as defining faults to investigate, injecting these faults into a MapReduce system, and analyzing the impact of these faults on the system performance. To do this, we have implemented MapReduce Benchmark Suite (MRBS) which is a fault injection tool devel-

Parameters	Values
Operating System	Centos 7.2, 64 bit
Hadoop Version	Hadoop-2.7.4
Hadoop Installation mode	Cluster
Master	1
Slaves	4
Hadoop replication factor	2 (is the number of times the block of data would get replicated)
CPU	3 cores
Memory	2 GB
Disk (SSD)	40 GB

Table 6.2 – *Parameters of Hadoop cluster*

oped by Sangroya et al [158].

MRBS [159] is a performance and dependability benchmark suite for MapReduce systems which includes a variety of benchmarks that covers a large range of application domains as well as execution scenarios. The tool generates automatic fault load and injects it in MapReduce environment at different rates. In addition, it provides a means to analyze the efficiency of fault-tolerance capability of Hadoop cluster. Figure 6.5 presents the process of fault tolerance evaluation using MRBS benchmark.

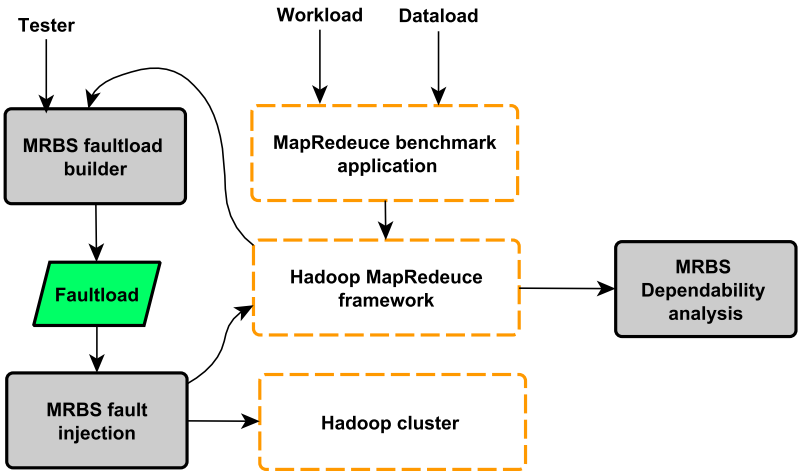


Figure 6.5 – *Fault tolerance evaluation with MRBS benchmark.*

Fault Injection: The injection of a fault in a MapReduce system with MRBS is treated differently depending on the type of fault considered. To mimic failures, we simply follow one of the following scenarios:

- A Node Crash Injection: To inject a node crash fault, we shut down a node or by using Linux kill command to terminate the TaskTracker and DataNode daemon processes

running on the node.

- A Task Process Crash Injection: To inject this fault, we kill he process running a task on a node.
- A Task Software Fault Injection: We implement this fault by generating an exception on a map task or a reduce task.

Workload: The workloads emulated by MRBS are selected to represent a range of loads, from the compute-intensive to the data-intensive. It consists of five benchmarks on different domains: *recommendation systems*, *business intelligence*, *bioinformatics*, *text processing*, and *data mining*. In our experiment we have used data-intensive *Text Processing* workload. The *Text Processing* consists of a standard application of MapReduce used, for example, to analyze the logs of search engines and websites [158]. A random generated text files of different sizes are used as input data of the benchmark.

In this chapter, two different scenarios have been simulated. The first one consists in analyzing the performance of Hadoop with presence of different types of failure (task failure, TaskTracker failure and NameNode failure) (subsection 6.4.3). The second one consists in investigating the impact of checkpointing interval selection on the performance of Hadoop under various failure probabilities (subsection 6.4.4).

6.4.3 The impact of failures on Hadoop performance

In this section, we evaluate the fault tolerance of Hadoop MapReduce by using MRBS benchmark. In the first experiment we run 5-nodes Hadoop cluster with Text Processing workload of size 10GB. We have used two MapReduce applications which are:

- **WordCount**: It is an application with a heavy Map phase and a light reduce phase [160]. Map outputs a $< word, 1 >$ key-value pair for each word in an input file. Reduce combines the count for each word producing a $< word, result >$ pair.
- **Sort**: It sorts the text input data [161]. A large amount of intermediate data are produced which leads to a heavy reduce phase.

Figure 6.6 presents job response time for WordCount and Sort jobs with injection of different failures. The x-axis represents different type of failures whereas the y-axis is the total response time of jobs. From Figure 6.6, we observe that the impact level on the response time of the MapReduce jobs depends on the type of failure. The "None" bars represent the case when MapReduce jobs run without injected failures, the WordCount

job spent about 970 second whereas the sort job took about 760 second. Concerning a "Kill Node" failure, one node on the cluster was stopped at runtime stage. When a node failure happens, the failed node will be removed from the cluster. We observe that the response time of WordCount job was not affected when a node failure occurs, which demonstrates that CPU intensive applications (WordCount) can be recovered by Hadoop with one node failure without any additional time. For the "Network-Slow" fault, network-packets were decelerated. For a task failure, we call "system.exit()" function on the application code. In the "Packet-Drop" fault, we drop some packets at many nodes. Task failure "system.exit()" has the worst impact on WordCount as shown in Figure 6.6. For Sort job, which is a network-intensive application, Network-Slow and Packet-Drop have an impact on job response time.

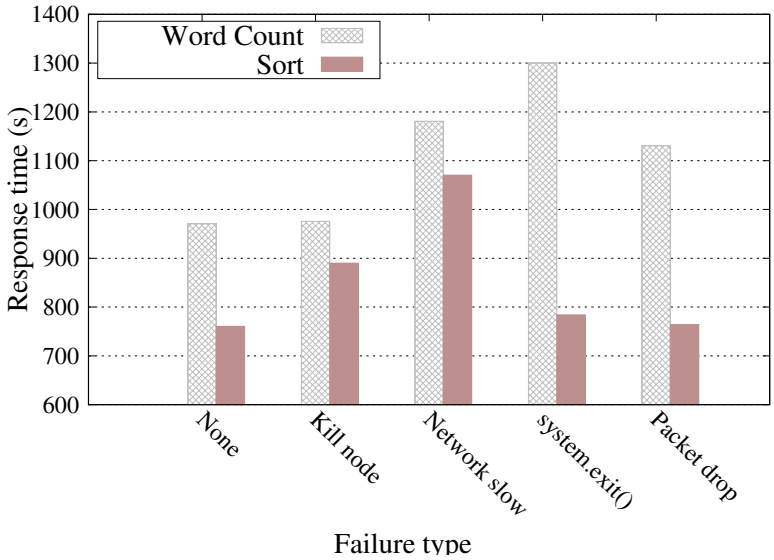


Figure 6.6 – *Job response time for Word Count and Sort under different failure types.*

In the second experiment we run 5-nodes Hadoop cluster using WordCount job with different sizes; for job 1 we use 1GB, for job 2 we use 5Gb and for Job 3 we use 10GB. Figure 6.7 presents Hadoop cluster availability with different number of node faults. The x-axis represents the number of killed nodes, whereas the y-axis is the percentage of operational time of the Hadoop cluster. To provide the availability of the Hadoop cluster with a percentage of 85%, Hadoop will allow only one node failure for the execution of job3, but Hadoop can allow until three node failures in case of the execution of job1 in a 5-nodes cluster. Briefly, if the size of input data of the MapReduce application is not large, Hadoop can treat faults with suitable availability level.

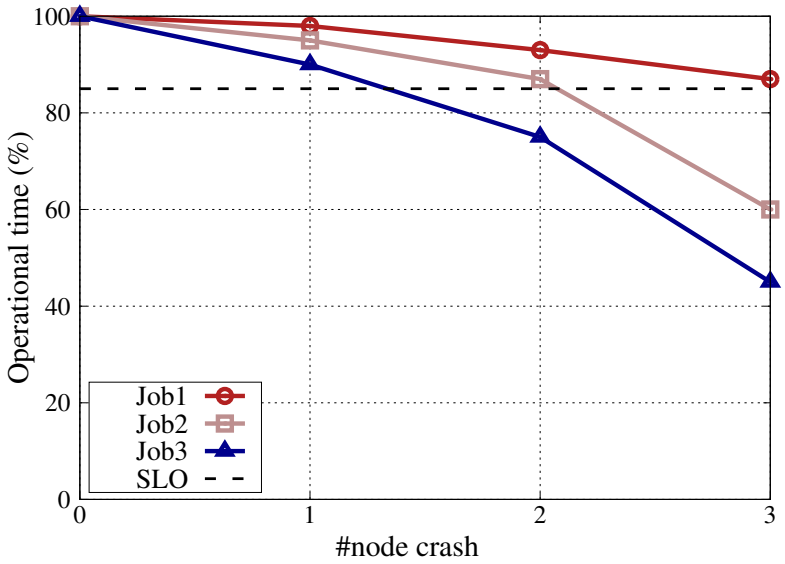


Figure 6.7 – *Hadoop cluster availability with different number of node faults.*

In the third experiment we run 5-node Hadoop cluster using WordCount job of size 10GB and we apply different split-sizes ranging from 16MB to 512MB. Figure 6.8 presents the average WordCount response time for different split-sizes under two task failure rates of 11% and 25%, and without failure. Figure 6.9 presents the slowdown or the additional

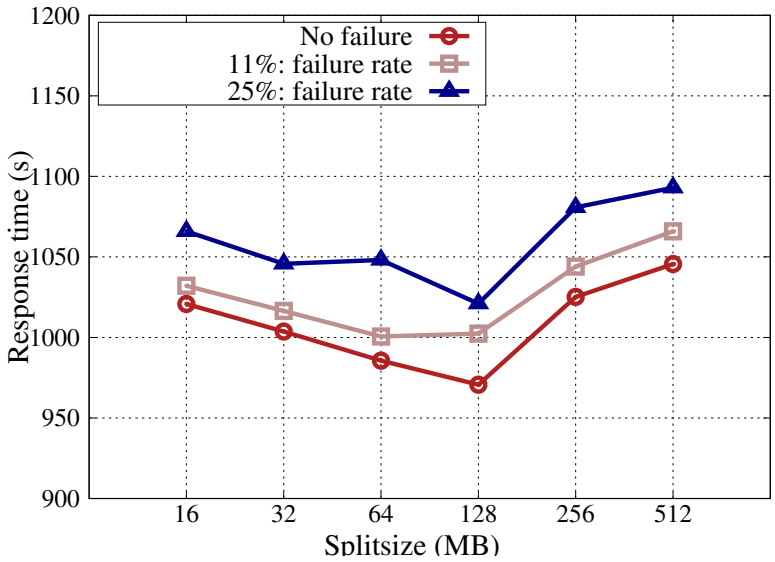


Figure 6.8 – *The impact of task failures on WordCount job*

execution time of WordCount under two task failure rates of 11% and 25%. From Figure 6.8, we see that when the task failure rate increases, the job response times increase too. The best job response time is achieved when the split-size equals 128MB with presence

and absence of failures. The reason of that is the block size equals the size of the default HDFS block used in our cluster (128MB). When the split size become larger than 128MB, the job response time increases. The reason of this is that the HDFS blocks that have a large size require an additional mappers to collect data that are generally situated on more than one node, hence increasing task execution time. In summary, from the results presented in Figures 6.8 and 6.9, we conclude that when the split-size is larger than the default HDFS bolck size, the same percentage of task failures could cause a greater impact on the task execution time since only few nodes will re-execute the failed tasks while many nodes remain unemployed.

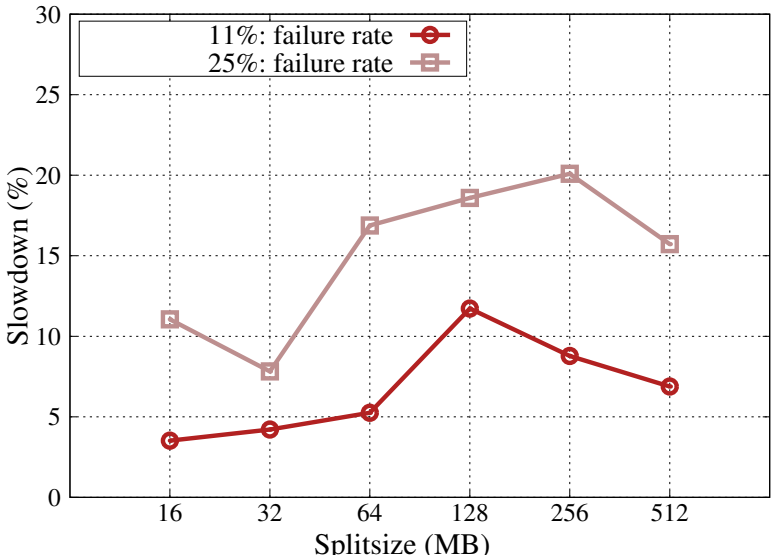


Figure 6.9 – *Slowdown percentage of WordCount job caused by task failures.*

6.4.4 The impact of checkpointing interval on Hadoop performance

In this section, we provide a further comprehensive performance analysis of Hadoop for an attempt to shed light on the relation between the checkpointing interval and failure probability with respect to the characteristics of the workload.

In this scenario, we evaluate both the total number of executed tasks that did not exceed its deadline as well as the results' precision of the executed tasks. Therefore, we have introduced a OSP metric, which is the overall system performance, and equals:

$$OSP = TGR * ARP \tag{6.3}$$

Where TGR is the guarantee ratio of the entire job. It is the average number of tasks, which are arrived at the central scheduler at the runtime stage, that meet their deadline. ARP represents the average precision results of the finished tasks. Here TGR is defined as:

$$TGR = \frac{T_G}{T_A} \tag{6.4}$$

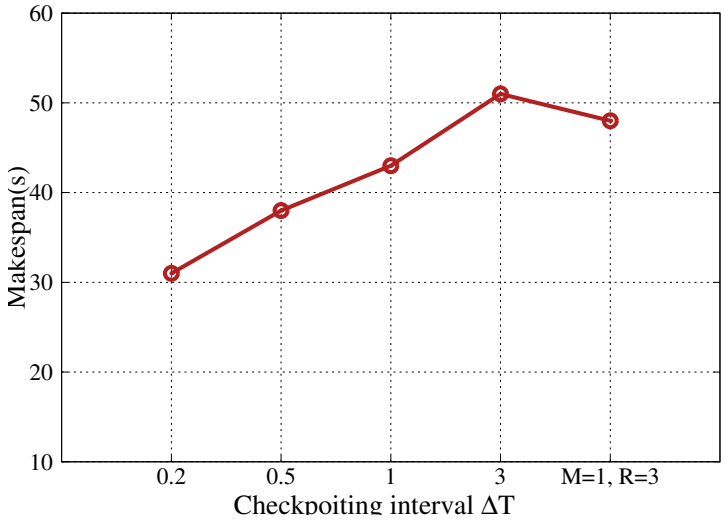
Where T_G is the total number of guaranteed tasks and T_A is the total number of tasks that arrived to the scheduler. According to the service level agreement (SLA) contract, it is supposed that both TGR and ARP are important factors and have almost the same impact for the OSP metric.

In the first simulation, we ran Wordcount program with failure probability FP=1 (a failure will be definitely occurred at runtime stage) in order to show the effect of checkpointing for decreasing the job execution time. Figure 6.10 shows that the makespan of the Wordcount program increases with the increase of checkpointing interval and vice versa. When the checkpointing interval equals $(\Delta T = 3)$, the Wordcount program experienced the largest makespan. This is due to the time delay caused by TaskTracker for sending information about failures or free slots to the JobTracker in case of a longer checkpointing interval. In other case, if the checkpointing interval is very small, the master node will be overloaded as demonstrated in our next simulation. In other words, when the checkpointing interval becomes shorter, the exchanged messages between the TaskTracker and the JobTracker become more frequent. Consequently, choosing the best checkpointing interval value is required to mitigate the load of the master node.

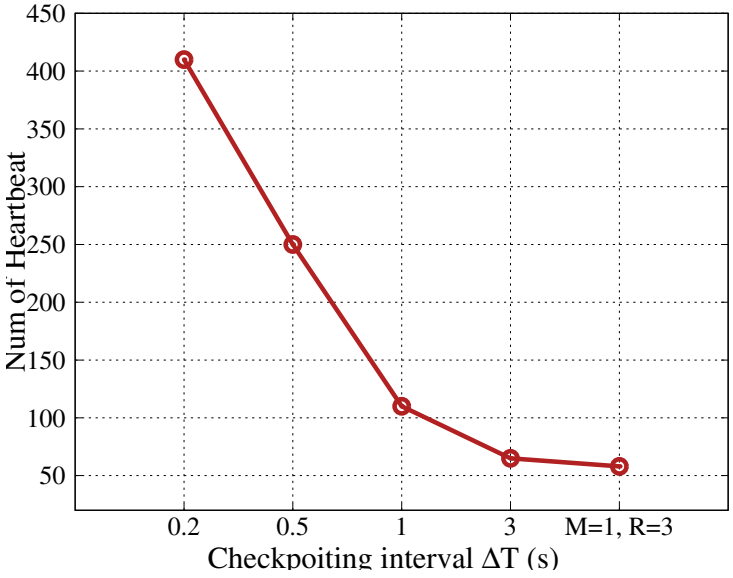
In Wordcount program, tasks spend the most of their execution time in the reduce phase. Hence, we should assign a big checkpointing interval to the reduce phase. We have adapted the checkpointing interval value at the runtime stage depending on map and reduce phases as shown Figure 6.10. Then, we have tested the Wordcount program with the following values (M=1, R =3), M for map and R for reduce. In this case, the Wordcount program has smaller makespan compared with the case when map and reduce have the same checkpointing interval value (M=3 and R =3).

In the second simulation, we use all the possible combinations of the following value of checkpointing intervals and failure probabilities:

- **CPI** = $\{0.1, 0.2, 0.3, 0.4, 0.5\}$
- **FP** = $\{0, 0.25, 0.5, 0.75, 1\}$



(a)



(b)

Figure 6.10 – *Makespan and messages number with different checkpointing intervals when executing the Wordcount application*

Figure 6.11 shows the performance of Hadoop MapReduce, with respect to the OSP indicator metric, under different combinations of failure probabilities and checkpointing intervals. It can be observed that large checkpointing intervals gave better performance when failure probabilities are low. However, when the failure probability increases, smaller checkpointing intervals yield a higher level of OSP. For all failure probabilities, the worst value of overall system performance is given when the checkpointing becomes more frequently which may increases the makespan of the jobs. Thus, this shows that

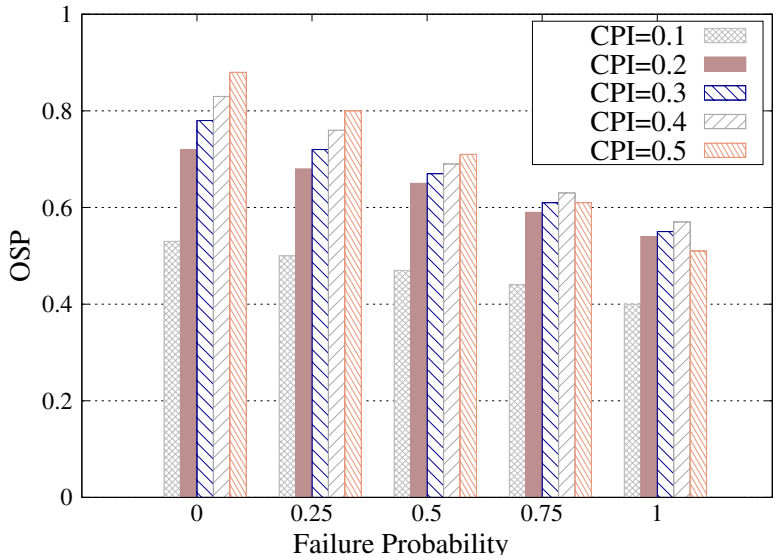


Figure 6.11 – *The overall system performance vs the failure probability (FP), for all of the investigated checkpointing intervals (CPI) In Hadoop.*

checkpointing does not need to be very frequent in order to achieve good performance even in the case where a failure always occurs at a runtime stage.

6.5 Conclusion

In distributed heterogeneous systems, failures are always present due to the large volume of processed data. Most of big companies and developers use Hadoop MapReduce, and, like other distributed systems, it needs to be tested against different types of failures. In this chapter, we exposed and analyzed Hadoop MapReduce fault tolerance mechanism through generating and executing representative fault cases. Then, we analyzed the relation between the checkpointing interval and failure probability, with respect to the characteristics of the workload based on the defined overall system performance (OSP) metric. We conducted our experiments based on MRBS benchmark that allows to characterize a fault load, generate it, and inject it in a Hadoop cluster. In summary, even though we did not experiment with a large amount of input data, from the obtained results, we observed that slowdowns are modest even with relatively high failure rate, which demonstrates that Hadoop provides reasonable resistance to failures. Furthermore, Hadoop has different behaviors to deal with failures depending on different parameters such as: *type of job, type of failure*, and also the *HDFS block size*. In addition, our experiments showed that the checkpointing interval value is an essential factor to

the task execution on Hadoop. In particular, for higher failure probabilities and smaller service times, checkpointing should be more frequent, in order to achieve good performance. However, unnecessary frequent checkpointing may degrade the system performance, therefore, the selected value of checkpointing interval should be above a particular threshold. In our future work, we intend to propose a new failure-aware scheduling strategy that predicts the occurrence probability of failures before assigning tasks to the virtual machines. This strategy aims at minimizing the recovery time of failed tasks, which can lead to decrease the execution time.

Overall conclusion and perspectives

Cloud computing has introduced a new business model of consuming the computing resources based on "pay as you go" which allows enterprises to pay only for the consumed resources. This model reduces business risks and maintenance expenses by outsourcing the service infrastructure to the cloud providers. On the other hand, the traditional model of software license acquisition deployed on the servers of the clients requires many operations such as updates and configurations, thus generating significant costs. Hence, cloud computing has emerged to reduce these additional costs and difficulties by executing tasks on centralized third-party servers and storage utilities. Cloud clients can use the computing resources without the management of the underlying infrastructure, which is the responsibility of the cloud providers. Thanks to these cloud benefits, enterprises have been migrating their applications towards cloud computing.

Clouds offer an environment for solving problems involving large-scale data. They provide on-request access to a virtually infinite pool of resources, which can be effectively arranged to suit diverse application needs. In addition, resources can be scaled in and out anytime to suit the present application needs and QoS prerequisites. However, there are a lot of issues and problems related to cloud computing that can slow its growth and even compromise its future. One of the main issues of cloud computing is to optimally schedule tasks on datacenters. While the cloud's popularity is increasing, tasks scheduling and resources allocation attract the attention of researchers to propose efficient scheduling algorithms. Generally, task scheduling is considered as assigning tasks to the available resources taking into account the characteristics and constraints of the tasks. Furthermore, cloud resources should be used efficiently without affecting cloud service parameters. Hence, cloud users are facing the problem of choosing the best resources to use.

This thesis tended to increment our knowledge about tasks scheduling problem and propose new strategies to optimize the scheduling and the allocation of resources in cloud computing. We have proposed three scheduling strategies, which contribute to identify the drawbacks of some scheduling algorithms and to introduce innovative techniques to deal with these issues. The first scheduling strategy balances the load between dat-

acenters and afterwards minimizes the overhead of data exchanges. The second strategy reduces the energy consumption and maximizes the efficiency of the available resources. The last strategy minimizes the makespan of a given workflow and maintains a well-balanced load across the virtual machines of the cluster. The first strategy has three phases, named respectively, *finding datasets dependencies phase, threshold assignment phase* and *load balancing phase*. The second strategy contains in its turn four procedures, named respectively, *tasks scheduling procedure, detection of physical machines state procedure, tasks selection procedure* and *tasks migration procedure*. The third strategy contains four phases, named respectively, *attribution of VMs threshold phase, datasets dependencies specification phase, task sorting phase* and *virtual machine selection phase*. Finally, we have addressed the fault tolerance mechanism of Hadoop MapReduce by analyzing the performance of Hadoop in the presence of different type of failures as well as investigating the impact of checkpointing interval selection on the performance of Hadoop.

In this thesis, we have simulated the previous three strategies using Cloudsim simulator to study their behaviors. In addition, we have compared the obtained results with the state-of-the-art algorithms for workflow scheduling such as genetic algorithm, HEFT (Heterogeneous Earliest Finish Algorithm). We have also compared our algorithms with other algorithms that are already implemented on Cloudsim simulator such as ThrMmt (Static Threshold and Minimum Migration Time) and UMC (Utilization and Minimum Correlation). As a performance metrics, we have used total data movements, index of load balancing, energy consumption, number of task migration, SLA violation and makespan. For testing the fault tolerance mechanism of Hadoop MapReduce, we have conducted our experiments based on MRBS benchmark using overall system performance (OSP), slowdown and response time as performance metrics.

Our proposed scheduling strategies allow to reduce the energy consumption of datacenters, minimize the execution time, decrease the overhead of data exchanges and maintains a well-balanced load among all machines on the cluster. Concerning the fault tolerance mechanism of Hadoop, we conclude that Hadoop provides a reasonable resistance to failures. In addition, Hadoop has different behaviors to deal with failures depending on different parameters, namely type of job, type of failure, and also the HDFS block size.

Following the work presented in this thesis, new research activities can be launched to improve our work. The perspectives that we propose can therefore be oriented towards the following directions:

- Regarding the scheduling of workflows based on matching game theory, we have only considered the energy consumed by the processors to compute the total energy consumption. We intend to generalize the model by taking into account other sources of energy consumption such as hard disks activities.
- In our proposed scheduling algorithm E-HEFT (Enhancement Heterogeneous Earliest Finish Time), we plan to consider the execution cost as another objective function for scheduling to become a multi-objective optimization problem.
- When a resource constraint occurs, a cloud provider may choose to delegate the tasks to other providers in order to avoid SLA violation and maintain the scalability, which is one of the most important aspect of cloud computing. This scenario lead to a new research direction in SLA management. In addition, the inter-clouds aspect may cause a problem for the implementation of our proposed algorithms in a distributed heterogeneous environment. Hence, we plan to extend our algorithms to inter-clouds.
- In the last chapter of our thesis, we have analyzed the behavior of Hadoop under different type of failures. Our ongoing work intend to extend this work by proposing a new model to determine the probability of failures before assigning tasks to virtual machines. Next, we will use our proposed algorithm E-HEFT (Enhancement Heterogeneous Earliest Finish Time) that takes into account the reliability of task execution while assigning tasks to virtual machines.

Appendix A

Cloudsim simulator

As the target system of our work is a generic cloud computing environment, it is essential to evaluate it on a large-scale virtualized infrastructure. However, performing benchmarking experiments in repeatable, dependable, and scalable environment using a real cloud platform would be very expensive and difficult, especially when it is necessary to reproduce the experiment with the same conditions to compare different algorithms. Therefore, a simulation has been chosen using CloudSim simulator as a way to evaluate our proposed algorithms.

A.1 Definition

CloudSim, project developed by the Melbourne CLOUDS Laboratory in Australia, is an extensible simulator which aims to enable modeling and simulation of cloud-based systems [110]. Particularly, CloudSim provides a generic broker modeled as a class named DataCenterBroker, we have extended this class to support DAG-structured workflows and to model the behavior of this component and its particular placement policies. In contrast to another simulation toolkits (e.g. SimGrid, GangSim), it permits the modeling of virtualized environments, supporting on demand resources provisioning and their management. In addition, it enables developers to focus only on the investigation of the specific system design issues, without worrying about the low level details related to the cloud infrastructures and services. About implementing Cloudsim, the VMs are considered as the cloud resources and Cloudlets as tasks/jobs [111].

A.2 CloudSim architecture

Figure A.1 presents a multi layered design of the CloudSim architecture as well as its software framework. The CloudSim architecture provides in the first layer (User Code) the modeling of the different entities of a cloud such as datacenter, physical machines and virtual machines. The next layer is the CloudSim which is made up of many fundamental classes developed in java such as Datacenter Broker, Datacenter characteristics, VMM Allocation Policy, Cloud Coordinator, Cloudlet Scheduler, VM Scheduler, Host, VM, etc. This layer manages the execution of the core entities (Virtual Machines, Hosts, Data centers, applications) during the simulation period.

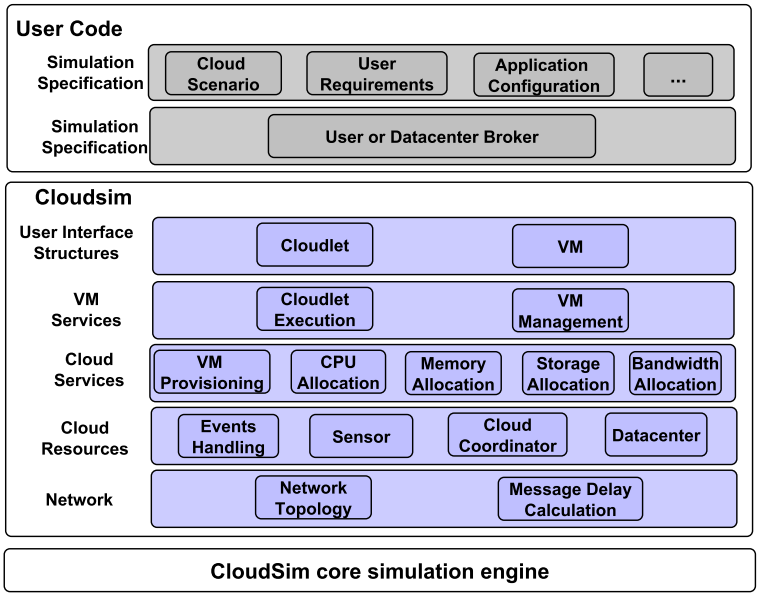


Figure A.1 – *CloudSim architecture [110]*

A.3 Cloud modeling

Each cloud is composed of a set of datacenters, which in their turn contains a set of hosts. In addition, each host is composed of a set of virtual machines. To simulate a given algorithms in a cloud computing environment, one class that contains the function Main() should be defined, with the parameters of the cloud's components such as the number of datacenter, hosts, VM and their characteristics like RAM, CPU, MIPS and

bandwidth [1]. The following scripts describe an example of the implementation of some cloud components in CloudSim simulator.

The configuration of the Virtual machine on this example is:

```
/*--VM description--*/
int vm_id = 1; //vm id
int vm_mips = 500; //Million of Intruction Per Second
long vm_size = 2500; //image size (MB)
int vm_ram = 1024; //vm memory (MB)
long vm_bw = 10000; //vm bandwidth (10 Mbit/s)
int vm_pes = 2; //number of cpus
String vmm = "Xen"; //VM monitor name

/*-- creation of  VMs--*/
Vm vm1 = new Vm(vm_id, brokerId, mips, vm_pes, vm_ram, vm_bw, vm_size, vmm
    , new CloudletSchedulerTimeShared());
```

And the configuration of the host is:

```
/*-- creation of hosts--*/
List<Host> host_List = new ArrayList<Host>();
List<Pe> pe_List = new ArrayList<Pe>();
int host_mips = 10000;
pe_List.add(new Pe(0, new PeProvisionerSimple(mips))); //need to store Pe
    id and MIPS Rating
int host_Id=0;
int host_ram = 4096; //host memory (MB)
long host_storage = 10000000; //host storage (10 GB)
int host_bw = 10000;

hostList.add(
new Host(
host_Id,
new RamProvisionerSimple(host_ram),
new BwProvisionerSimple(host_bw),
host_storage,
pe_List,
new VmSchedulerSpaceShared(pe_List)
)
);
```

[1] https://graal.ens-lyon.fr/~ecaron/m2rts/2015/blogbazm/

And finally the configuration of the datacenter:

```
32 /*-- Create datacenter --*/
33 String arch = "x86"; // system architecture
34 String os = "Linux"; // operating system
35 String vmm = "Xen";
36 double time_zone = 10.0; // the time zone
37 double cost = 3.0; // the cost of  processing
38 double costPerMem = 0.05; // the cost of ram memory
39 double costPerStorage = 0.001; // the cost of storage
40 double costPerBw = 0.0; // the cost of bandwidth
41 LinkedList<Storage> storage_List = new LinkedList<Storage>(); //without
       adding SAN devices by now
42
43 DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
        arch, os, vmm, host_List, time_zone, cost, costPerMem, costPerStorage
       , costPerBw);
44
45 Datacenter datacenter = null;
46 try {
47 datacenter = new Datacenter(name, characteristics, new
       VmAllocationPolicySimple(host_List), storage_List, 0);
48 } catch (Exception e) {
49 e.printStackTrace();
50 }
```

List of publications

International Journals

1. **Yassir Samadi**, Mostapha Zbakh and Claude Tadonki. "Performance comparison between Hadoop and Spark frameworks using HiBench benchmarks". Concurrency and Computation: Practice and Experience. 2018, vol. 30, no 12, p. e4367. https://doi.org/10.1002/cpe.4367
2. **Yassir Samadi**, Mostapha Zbakh and Claude Tadonki. "Graph-based Model and Algorithm for Minimizing Big Data Movement in a Cloud Environment". International Journal of High Performance Computing and Networking. 2018, Inderscience, in press.
3. **Yassir Samadi**, Mostapha Zbakh & Claude Tadonki (2018): "DT-MG: many- to-one matching game for tasks scheduling towards resources optimization in cloud computing". International Journal of Computers and Applications. https://doi.org/10.1080/1206212X.2018.1519630

International Conferences

4. **Yassir Samadi**, Mostapha Zbakh and Claude Tadonki. "Comparative study between Hadoop and Spark based on Hibench benchmarks". In Cloud Computing Technologies and Applications (CloudTech), 2nd International Conference on (pp. 267-275). IEEE. May 2016, Merrakech, Morocco.
5. **Yassir Samadi** and Mostapha Zbakh and Claude Tadonki. "Threshold-based load balancing algorithm for Big Data on a Cloud environment". In Proceedings of the 2nd international Conference on Big Data, Cloud and Applications, 2017, March, p. 18). ACM. Tetuan, Morocco.
6. Amin Haouari, Mostapha Zbakh, Rachid Cherkaoui, **Yassir Samadi**, and Najlae Kasmi. (2017, October). "TASMR: Towards advanced secure mapreduc framework across untrusted hybrid clouds". In Cloud Computing Technologies and Applications (CloudTech), 2017 3rd International Conference of (pp. 1-9). IEEE. Rabat, Morocco.

7. Najlae Kasmi, Mostapha Zbakh, **Yassir Samadi**, Rachid Cherkaoui." Performance evaluation of StarPU schedulers with preconditioned conjugate gradient solver on heterogeneous (multi-CPUs/multi-GPUs) architecture". In proceedings of the 3rd International Conference on Cloud Computing Technologies and Applications (Cloud-Tech). pp. 1-6. IEEE. Oct, 2017. Rabat, Morocco.

8. **Yassir Samadi** and Mostapha Zbakh. "The impact of checkpointing interval selection on the scheduling performance of Hadoop framework". 2018 6th International Conference on Multimedia Computing and Systems (ICMCS). IEEE, 2018. Rabat, Morocco.

9. **Yassir Samadi**, Mostapha Zbakh and Claude Tadonki. "E-HEFT: Enhancement Heterogeneous Earliest Finish Time algorithm for Task Scheduling based on Load Balancing in Cloud Computing". 2018 International Conference on High Performance Computing & Simulation (HPCS). IEEE, 2018. Orléans, France.

10. **Yassir Samadi**, Mostapha Zbakh and Claude Tadonki. "Analyzing fault tolerance mechanism of Hadoop Mapreduce under different type of failures". 2018 Cloud Computing Technologies and Applications (CloudTech 18) IEEE. Brussels, Belgium.

11. **Yassir Samadi**, Mostapha Zbakh and Claude Tadonki."Energy-efficient and task migration approach in cloud datacenters". Submitted.

BOOK CHAPTERS

12. **Yassir Samadi**, Mostapha Zbakh and Amin Haouari. (2018). "Big Data Processing on Cloud Computing Using Hadoop Mapreduce and Apache Spark". In K. Munir (Ed.), Cloud Computing Technologies for Green Enterprises (pp. 224-250). Hershey, PA: IGI Global. doi:10.4018/978-1-5225-3038-1.ch009.

13. Amin Haouari, Mostapha Zbakh and **Yassir Samadi**. (2018). "Current State Survey and Future Opportunities for Trust and Security in Green Cloud Computing". In K. Munir (Ed.), Cloud Computing Technologies for Green Enterprises (pp. 83-113). Hershey, PA: IGI Global. DOI: 10.4018/978-1-5225-3038-1.ch004.

14. **Yassir Samadi**, Mostapha Zbakh and Claude Tadonki. "Workflow Scheduling Issues and Techniques in Cloud Computing: A Systematic Literature Review". Springer International Publishing. Cloud Computing and Big Data: Technologies, Applications and Security, 49, pp.241-263, 2018.

Bibliography

[1] G. Amir and H. Murtaza, "Beyond the hype: Big data concepts, methods, and analytics," *International Journal of Information Management*, vol. 35, no. 2, pp. 137–144, 2015.

[2] P. Senkul and T. I. Hakki, "An architecture for workflow scheduling under resource allocation constraints," *International Journal of Information Management*, vol. 30, no. 5, pp. 399–422, 2005.

[3] B. Simion, C. Leordeanu, F. Pop, and V. Cristea, "A hybrid algorithm for scheduling workflow applications in grid environments (icpdp)," in *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems", Berlin, Heidelberg*. Springer, November 2007, pp. 1331–1348.

[4] V. Mihaela-Andreea, P. Florin, T. Radu-Ioan, C. Valentin, and K. Joanna, "Resource-aware hybrid scheduling algorithm in heterogeneous distributed computing," *Future Generation Computer Systems*, vol. 51, pp. 61–71, 2017.

[5] E. Sina, P. Ali, and G. Maziar, "Structure-aware online virtual machine consolidation for datacenter energy improvement in cloud computing," *Computers & Electrical Engineering*, vol. 42, pp. 74–89, 2015.

[6] J. Si-Yuan, A. Shahzad, S. Kun, and Z. Yi, "State-of-the-art research study for green cloud computing," *The Journal of Supercomputing*, vol. 65, no. 1, pp. 445–468, 2013.

[7] Z. Yong, C. Liang, L. Youfu, and T. Wenhong, "Efficient task scheduling for many task computing with resource attribute selection," *China Communications*, vol. 11, no. 12, pp. 125–140, 2014.

[8] B. Peter and P. Brucker, *Scheduling algorithms*. Springer, 2007, vol. 3.

[9] M. Maciej, F. Kamil, B. Marian, D. Ewa, and N. Jarek, "Scheduling multilevel deadline-constrained scientific workflows on clouds based on cost optimization," *Scientific Programming*, vol. 2015, p. 5, 2015.

[10] N. SH, O. SK, and N. AYC, "An improved intelligent water drops algorithm for achieving optimal job-shop scheduling solutions," *International Journal of Production Research*, vol. 50, no. 15, pp. 4192–4205, 2012.

[11] K. M. Sudan, G. Indrajeet, and J. P. K, "Forward load aware scheduling for data-intensive workflow applications in cloud system," in *Information Technology (ICIT), 2016 International Conference on*. IEEE, 2016, pp. 93–98.

[12] Y. Sonia, C. Rachid, K. Hubert, and G. Bertrand, "Multi-objective approach for energy-aware workflow scheduling in cloud computing environments," *The Scientific World Journal*, vol. 2013, 2013.

[13] A. AR, M. D, and S. Vijayan, "Ffbat: A security and cost-aware workflow scheduling approach combining firefly and bat algorithms," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 24, 2017.

[14] W. Wei-Jen, C. Yue-Shan, L. Win-Tsung, and L. Yi-Kang, "Adaptive scheduling for parallel tasks with qos satisfaction for hybrid cloud environments," *The Journal of Supercomputing*, vol. 66, no. 2, pp. 783–811, 2017.

[15] V. L. M, R.-M. Luis, C. Juan, and L. Maik, "A break in the clouds: towards a cloud definition," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 50–55, 2008.

[16] R. B. Prasad and C. Eunmi, "A service-oriented taxonomical spectrum, cloudy challenges and opportunities of cloud computing," *International Journal of Communication Systems*, vol. 25, no. 6, pp. 796–819, 2012.

[17] C. Huangke, Z. Xiaomin, Q. Dishan, L. Ling, and D. Zhihui, "Scheduling for workflows with security-sensitive intermediate data by selective tasks duplication in clouds," *IEEE transactions on Parallel and Distributed Systems*, vol. 28, no. 9, pp. 2674–2688, 2017.

[18] W. Cong, R. Kui, and W. Jia, "Secure and practical outsourcing of linear programming in cloud computing," in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 820–828.

[19] W. Lifei, Z. Haojin, C. Zhenfu, D. Xiaolei, J. Weiwei, C. Yunlu, and V. A. V, "Security and privacy for storage and computation in cloud computing," *Information Sciences*, vol. 258, pp. 371–386, 2014.

[20] W. Marek, H. Andreas, and P. Radu, "Towards a general model of the multi-criteria workflow scheduling on the grid," *Future Generation Computer Systems*, vol. 25, no. 3, pp. 237–256, 2009.

[21] B. Kitchenham and S. Charters, "Technical report: Guidelines for performing systematic literature reviews in software engineering," 2007, (Date last accessed 15-July-2017). [Online]. Available: https://userpages.uni-koblenz.de/~laemmel/esecourse/slides/slr.pdf

[22] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering–a systematic literature review," *Information and software technology*, vol. 51, no. 1, pp. 7–15, 2009.

[23] P. Mell, T. Grance *et al.*, "The nist definition of cloud computing," 2011.

[24] S. Sharma, "Expanded cloud plumes hiding big data ecosystem," *Future Generation Computer Systems*, vol. 59, pp. 63–92, 2016.

[25] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of internet services and applications*, vol. 1, no. 1, pp. 7–18, 2010.

[26] A. Zahariev, "Google app engine," *Helsinki University of Technology*, pp. 1–5, 2009.

[27] G. L. Stavrinides and H. D. Karatza, "A cost-effective and qos-aware approach to scheduling real-time workflow applications in paas and saas clouds," in *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*. IEEE, 2016, pp. 231–239.

[28] H. Jin, S. Ibrahim, T. Bell, W. Gao, D. Huang, and S. Wu, "Cloud types and services," in *Handbook of Cloud Computing*. Springer, 2010, pp. 335–355.

[29] Y. Du and X. Li, "Application of workflow technology to current dispatching order system," *International Journal of Computer Science and Network Security*, vol. 8, no. 3, pp. 59–61, 2008.

[30] W. Standard-Interoperability, "Workflow management coalition workflow standard-interoperability wf-xml binding," 1999.

[31] D. M. Abdelkader and F. Omara, "Dynamic task scheduling algorithm with load balancing for heterogeneous computing system," *Egyptian Informatics Journal*, vol. 13, no. 2, pp. 135–145, 2012.

[32] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-science: An overview of workflow system features and capabilities," *Future generation computer systems*, vol. 25, no. 5, pp. 528–540, 2009.

[33] G. B. Berriman, E. Deelman, J. C. Good, J. C. Jacob, D. S. Katz, C. Kesselman, A. C. Laity, T. A. Prince, G. Singh, and M.-H. Su, "Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand," in *Optimizing Scientific Return for Astronomy through Information Technologies*, vol. 5493. International Society for Optics and Photonics, 2004, pp. 221–233.

[34] R. Graves, T. H. Jordan, S. Callaghan, E. Deelman, E. Field, G. Juve, C. Kesselman, P. Maechling, G. Mehta, K. Milner *et al.*, "Cybershake: A physics-based seismic hazard model for southern california," *Pure and Applied Geophysics*, vol. 168, no. 3-4, pp. 367–381, 2011.

[35] C.-x. YE and J. LU, "Igrid task scheduling based on improved genetic algorithm,"," *Computer Science*, vol. 37, no. 7, pp. 233–235, 2007.

[36] L. F. Bittencourt and E. R. M. Madeira, "Hcoc: a cost optimization algorithm for workflow scheduling in hybrid clouds," *Journal of Internet Services and Applications*, vol. 2, no. 3, pp. 207–227, 2011.

[37] L. Zeng, B. Veeravalli, and X. Li, "Saba: A security-aware and budget-aware workflow scheduling strategy in clouds," *Journal of parallel and Distributed computing*, vol. 75, pp. 141–151, 2015.

[38] W. Zheng and R. Sakellariou, "Budget-deadline constrained workflow planning for admission control," *Journal of grid computing*, vol. 11, no. 4, pp. 633–651, 2016.

[39] L. Zhao, Y. Ren, and K. Sakurai, "Reliable workflow scheduling with less resource redundancy," *Parallel Computing*, vol. 39, no. 10, pp. 567–585, 2013.

[40] X. Wang, C. S. Yeo, R. Buyya, and J. Su, "Optimizing the makespan and reliability for workflow applications with reputation and a look-ahead genetic algorithm," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1124–1134, 2011.

[41] C. Zhang, E.-C. Chang, and R. H. Yap, "Tagged-mapreduce: A general framework for secure computing with mixed-sensitivity data on hybrid clouds," in *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*. IEEE, 2014, pp. 31–40.

[42] H. Topcuoglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE transactions on parallel and distributed systems*, vol. 13, no. 3, pp. 260–274, 2002.

[43] W. Tan, Y. Sun, L. X. Li, G. Lu, and T. Wang, "A trust service-oriented scheduling model for workflow applications in cloud computing," *IEEE Systems Journal*, vol. 8, no. 3, pp. 868–878, 2014.

[44] L. Liu, M. Zhang, R. Buyya, and Q. Fan, "Deadline-constrained coevolutionary genetic algorithm for scientific workflow scheduling in cloud computing," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 5, 2017.

[45] W. Chen, R. F. da Silva, E. Deelman, and R. Sakellariou, "Using imbalance metrics to optimize task clustering in scientific workflow executions," *Future Generation Computer Systems*, vol. 46, pp. 69–84, 2015.

[46] I. Casas, J. Taheri, R. Ranjan, L. Wang, and A. Y. Zomaya, "A balanced scheduler with data reuse and replication for scientific workflows in cloud computing systems," *Future Generation Computer Systems*, vol. 74, pp. 168–178, 2017.

[47] D. Poola, K. Ramamohanarao, and R. Buyya, "Enhancing reliability of workflow execution using task replication and spot instances," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 10, no. 4, p. 30, 2016.

[48] N. Rehani and R. Garg, "Reliability-aware workflow scheduling using monte carlo failure estimation in cloud," in *Proceedings of International Conference on Communication and Networks*. Springer, 2017, pp. 139–153.

[49] G. Xie, G. Zeng, Y. Chen, Y. Bai, Z. Zhou, R. Li, and K. Li, "Minimizing redundancy to satisfy reliability requirement for a parallel application on heterogeneous service-oriented systems," *IEEE Transactions on Services Computing*, 2017.

[50] S. Wang, Z. Liu, Z. Zheng, Q. Sun, and F. Yang, "Particle swarm optimization for energy-aware virtual machine placement optimization in virtualized data centers,"

in *Parallel and Distributed Systems (ICPADS), 2013 International Conference on*. IEEE, 2017, pp. 102–109.

[51] H. Duan, C. Chen, G. Min, and Y. Wu, "Energy-aware scheduling of virtual machines in heterogeneous cloud computing systems," *Future Generation Computer Systems*, vol. 74, pp. 142–150, 2017.

[52] Z. Li, J. Ge, H. Yang, L. Huang, H. Hu, H. Hu, and B. Luo, "A security and cost aware scheduling algorithm for heterogeneous tasks of scientific workflow in clouds," *Future Generation Computer Systems*, vol. 65, pp. 140–152, 2016.

[53] A. G. García, I. B. Espert, and V. H. García, "Sla-driven dynamic cloud resource management," *Future Generation Computer Systems*, vol. 31, pp. 1–11, 2014.

[54] S. K. Garg, A. N. Toosi, S. K. Gopalaiyengar, and R. Buyya, "Sla-based virtual machine management for heterogeneous workloads in a cloud datacenter," *Journal of Network and Computer Applications*, vol. 45, pp. 108–120, 2014.

[55] H.-Z. Zhou, K.-C. Huang, and F.-J. Wang, "Dynamic resource provisioning for interactive workflow applications on cloud computing platform," in *Russia-Taiwan Symposium on Methods and Tools of Parallel Processing*. Springer, 2010, pp. 115–125.

[56] S. J. Ludtke, P. R. Baldwin, and W. Chiu, "Eman: semiautomated software for high-resolution single-particle reconstructions," *Journal of structural biology*, vol. 128, no. 1, pp. 82–97, 1999.

[57] A. Mandal, K. Kennedy, C. Koelbel, G. Marin, J. M. Mellor-Crummey, B. Liu, and S. L. Johnsson, "Scheduling strategies for mapping application workflows onto the grid." in *hpdc*, vol. 5, 2005, pp. 125–134.

[58] Y. Samadi, M. Zbakh, and C. Tadonki, "Performance comparison between hadoop and spark frameworks using hibench benchmarks," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 12, p. e4367, 2017.

[59] J. Dean and G. Sanjay, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[60] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'10.

Berkeley, CA, USA: USENIX Association, 2010, pp. 10–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1863103.1863113

[61] G. Caragnano, L. Mossucca, M. A. Francavilla, P. Ruiu, and O. Terzo, "Design and implementation of a cloud computing platform for electromagnetic modelling," *International Journal of High Performance Computing and Networking*, vol. 8, no. 3, pp. 248–258, 2015.

[62] F. Etro, "The economic impact of cloud computing on business creation, employment and output in europe," *Review of Business and Economics*, vol. 54, no. 2, pp. 179–208, 2009.

[63] Z. Lu, J. Wu, J. Bao, and P. C. Hung, "Ocrem: Openstack-based cloud datacentre resource monitoring and management scheme," *International Journal of High Performance Computing and Networking*, vol. 9, no. 1-2, pp. 31–44, 2016.

[64] Ü. V. Çatalyürek, K. Kaya, and B. Uçar, "Integrated data placement and task assignment for scientific workflows in clouds," in *Proceedings of the fourth international workshop on Data-intensive distributed computing*. ACM, 2011, pp. 45–54.

[65] P. Nawrocki and W. Reszelewski, "Resource usage optimization in mobile cloud computing," *Computer Communications*, vol. 99, pp. 1–12, 2017.

[66] W. Hussein, T. Peng, and G. Wang, "A weighted throttled load balancing approach for virtual machines in cloud environment," *International Journal of Computational Science and Engineering*, vol. 11, no. 4, pp. 402–408, 2015.

[67] E. Deelman and A. Chervenak, "Data management challenges of data-intensive scientific workflows," in *Cluster Computing and the Grid, 2008. CCGRID'08. 8th IEEE International Symposium on*. IEEE, 2008, pp. 687–692.

[68] Y. Samadi, M. Zbakh, and C. Tadonki, "Graph-based model and algorithm for minimizing big data movement in a cloud environment," *International Journal of High Performance Computing and Networking*, 2017.

[69] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, and X. Qin, "Improving mapreduce performance through data placement in heterogeneous hadoop clusters," in *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–9.

[70] M. Wang, J. Zhang, F. Dong, and J. Luo, "Data placement and task scheduling optimization for data intensive scientific workflow in multiple data centers environment," in *Advanced Cloud and Big Data (CBD), 2014 Second International Conference on*. IEEE, 2014, pp. 77–84.

[71] K. Deng, K. Ren, J. Song, D. Yuan, Y. Xiang, and J. Chen, "A clustering based coscheduling strategy for efficient scientific workflow execution in cloud computing," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 18, pp. 2523–2539, 2013.

[72] Z. Er-Dun, Q. Yong-Qiang, X. Xing-Xing, and C. Yi, "A data placement strategy based on genetic algorithm for scientific workflows," in *Computational Intelligence and Security (CIS), 2012 Eighth International Conference on*. IEEE, 2012, pp. 146–149.

[73] D. Yuan, Y. Yang, X. Liu, and J. Chen, "A data placement strategy in scientific cloud workflows," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1200–1214, 2010.

[74] P. Teli, M. V. Thomas, and K. Chandrasekaran, "Big data migration between data centers in online cloud environment," *Procedia Technology*, vol. 24, pp. 1558–1565, 2016.

[75] L. Zeng, B. Veeravalli, and A. Y. Zomaya, "An integrated task computation and data management scheduling strategy for workflow applications in cloud environments," *Journal of Network and Computer Applications*, vol. 50, pp. 39–48, 2015.

[76] S. Saini, R. Ciotti, B. T. Gunney, T. E. Spelce, A. Koniges, D. Dossa, P. Adamidis, R. Rabenseifner, S. R. Tiyyagura, and M. Mueller, "Performance evaluation of supercomputers using hpcc and imb benchmarks," *Journal of Computer and System Sciences*, vol. 74, no. 6, pp. 965–982, 2008.

[77] C.-T. Yang and L.-H. Cheng, "Implementation of a performance-based loop scheduling on heterogeneous clusters," in *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 2009, pp. 44–54.

[78] D. Zhang, Y. Shou, and J. Xu, "An improved parallel k-means algorithm based on mapreduce," *International Journal of Embedded Systems*, vol. 9, no. 3, pp. 275–282, 2017.

[79] P. Teli, M. V. Thomas, and K. Chandrasekaran, "An efficient approach for cost optimization of the movement of big data," *Open Journal of Big Data (OJBD)*, vol. 1, no. 1, pp. 4–15, 2015.

[80] D. B. LD and P. V. Krishna, "Honey bee behavior inspired load balancing of tasks in cloud computing environments," *Applied Soft Computing*, vol. 13, no. 5, pp. 2292–2303, 2013.

[81] J. Jiahui, L. Junzhou, S. Aibo, D. Fang, and X. Runqun, "Bar: An efficient data locality driven task scheduling algorithm for cloud computing," in *Cluster, Cloud and Grid Computing*. IEEE, 2011, pp. 295–304.

[82] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang, "Mars: a mapreduce framework on graphics processors," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. ACM, 2008, pp. 260–269.

[83] F. Chang, J. Ren, and R. Viswanathan, "Optimal resource allocation for batch testing," in *Software Testing Verification and Validation, 2009. ICST'09. International Conference on*. IEEE, 2009, pp. 91–100.

[84] F. Changg, J. Ren, and R. Viswanathan, "Optimal resource allocation in clouds," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. IEEE, 2010, pp. 418–425.

[85] Q.-y. Huang and T.-l. Huang, "An optimistic job scheduling strategy based on qos for cloud computing," in *Intelligent Computing and Integrated Systems (ICISS), 2010 International Conference on*. IEEE, 2010, pp. 673–675.

[86] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," in *Proceedings of the 2008 conference on Power aware computing and systems*, vol. 10. San Diego, California, 2008, pp. 1–5.

[87] K. H. Kim, A. Beloglazov, and R. Buyya, "Power-aware provisioning of virtual machines for real-time cloud services," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 13, pp. 1491–1505, 2011.

[88] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future generation computer systems*, vol. 28, no. 5, pp. 755–768, 2012.

[89] Y. C. Lee and A. Y. Zomaya, "Energy efficient utilization of resources in cloud computing systems," *The Journal of Supercomputing*, vol. 60, no. 2, pp. 268–280, 2012.

[90] Z. Xiao, J. Jiang, Y. Zhu, Z. Ming, S. Zhong, and S. Cai, "A solution of dynamic vms placement problem for energy consumption optimization based on evolutionary game theory," *Journal of Systems and Software*, vol. 101, pp. 260–272, 2015.

[91] J. Torres, D. Carrera, K. Hogan, R. Gavaldà, V. Beltran, and N. Poggi, "Reducing wasted resources to help achieve green data centers," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*. IEEE, 2008, pp. 1–8.

[92] C.-H. Hsu, K. D. Slagter, S.-C. Chen, and Y.-C. Chung, "Optimizing energy consumption with task consolidation in clouds," *Information Sciences*, vol. 258, pp. 452–462, 2014.

[93] Y. Samadi, M. Zbakh, and C. Tadonki, "Dt-mg: many- toone matching game for tasks scheduling towards resources optimization in cloud computing," *International Journal of Computers and Applications*, pp. 1–13, 2018.

[94] O. Oualhaj, A. Kobbane, S. Essaid, and J. Ben-Othman, "A coalitional-game-based incentive mechanism for content caching in heterogeneous delay tolerant networks," *In International Conference In Wireless Communications and Mobile Computing Conference (IWCMC)*, pp. 987–992, 2015.

[95] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *The American Mathematical Monthly*, vol. 69, no. 1, pp. 9–15, 1962.

[96] A. E. Roth, "The college admissions problem is not equivalent to the marriage problem," *Journal of economic Theory*, vol. 36, no. 2, pp. 277–288, 1985.

[97] R. E. Alvin, "Deferred acceptance algorithms: History, theory, practice, and open questions," *international Journal of game Theory*, vol. 36, no. 3-4, pp. 537–569, 2008.

[98] R. Buyya, A. Beloglazov, and J. Abawajy, "Energy-efficient management of data center resources for cloud computing: a vision, architectural elements, and open challenges," *In Proceedings of the 2010 International Conference on Parallel and Distributed Processing Techniques and Applications*, 2010.

[99] Z. Dong, N. Liu, and R. Rojas-Cessa, "Greedy scheduling of tasks with time constraints for energy-efficient cloud-computing data centers," *Journal of Cloud Computing*, vol. 4, no. 1, p. 5, 2015.

[100] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, p. 599âĂŞ616, 2009.

[101] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, 2012.

[102] A. E. Roth and M. A. Oliveira Sotomayor, *Two-sided matching : a study in game-theoretic modeling and analysis*, ser. Econometric society monographs. Cambridge, Mass.: Cambridge university press, 1992, 1990.

[103] S. Khan and A. Ishfaq, "A cooperative game theoretical technique for joint optimization of energy consumption and response time in computational grids," *IEEE Trans Parallel Distrib Syst*, vol. 21, no. 4, pp. 537âĂŞ–553, 2009.

[104] A. Beloglazov and R. Buyya, "Energy efficient resource management in virtualized cloud data centers," in *Proceedings of the 2010 10th IEEE/ACM international conference on cluster, cloud and grid computing*. IEEE Computer Society, 2010, pp. 826–831.

[105] C.-H. Lien, M. F. Liu, Y.-W. Bai, C. H. Lin, and M.-B. Lin, "Measurement by the software design for the power consumption of streaming media servers," in *Instrumentation and Measurement Technology Conference, 2006. IMTC 2006. Proceedings of the IEEE*. IEEE, 2006, pp. 1597–1602.

[106] C.-H. Hsu, S.-C. Chen, C.-C. Lee, H.-Y. Chang, K.-C. Lai, K.-C. Li, and C. Rong, "Energy-aware task consolidation technique for cloud computing," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*. IEEE, 2011, pp. 115–121.

[107] L. Gkatzikis and I. Koutsopoulos, "Migrate or not? exploiting dynamic task migration in mobile cloud computing systems," *IEEE Wireless Communications*, vol. 20, no. 3, pp. 24–32, 2013.

[108] C. Bo, C. Jiming, Y. Zaiyue, and Y. Zhang, "Demand response management with multiple utility companies: a two level game approach," *IEEE Trans Smart Grid*, vol. 5, no. 2, pp. 722–731, 2014.

[109] H. Khani, N. Yazdani, and S. Mohammadi, "A self-organized load balancing mechanism for cloud computing," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 4, 2017.

[110] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

[111] R. N. Calheiros, R. Ranjan, C. A. De Rose, and R. Buyya, "Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services," *arXiv preprint arXiv:0903.2525*, 2009.

[112] M. A. H. Monil and R. M. Rahman, "Vm consolidation approach based on heuristics fuzzy logic, and migration control," *Journal of Cloud Computing*, vol. 5, no. 1, pp. 1–18, 2016.

[113] K. Park and V. S. Pai, "Comon: a mostly-scalable monitoring system for planetlab," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 1, pp. 65–74, 2006.

[114] A. Beloglazov and R. Buyya, "Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers." in *MGC@ Middleware*, 2010, p. 4.

[115] X. Fu and C. Zhou, "Virtual machine selection and placement for dynamic consolidation in cloud computing environment." *Frontiers of Computer Science*, vol. 9, no. 2, pp. 322–330, 2015.

[116] Z. Cao and S. Dong, "Dynamic vm consolidation for energy-aware and sla violation reduction in cloud computing," in *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2012 13th International Conference on*. IEEE, 2012, pp. 363–369.

[117] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: The laws of diminishing returns," 2010.

[118] S. M. Ghafari, M. Fazeli, A. Patooghy, and L. Rikhtechi, "Bee-mmt: A load balancing method for power consumption management in cloud computing," in *Contemporary Computing (IC3), 2013 Sixth International Conference on*. IEEE, 2013, pp. 76–80.

[119] A. Horri, M. Sadegh Mozafari, and G. Dastghaibyfard, "Novel resource allocation algorithms to performance and energy efficiency in cloud computing," *J Supercomput*, vol. 69, no. 3, pp. 1445–1461, 2014.

[120] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, and M. Livny, "Pegasus: Mapping scientific workflows onto the grid," in *Grid Computing*. Springer, 2004, pp. 11–20.

[121] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the kepler system," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[122] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat *et al.*, "Taverna: a tool for the composition and enactment of bioinformatics workflows," *Bioinformatics*, vol. 20, no. 17, pp. 3045–3054, 2004.

[123] J. Yu and R. Buyya, "A taxonomy of workflow management systems for grid computing," *Journal of Grid Computing*, vol. 3, no. 3-4, pp. 171–200, 2005.

[124] L. F. Bittencourt, R. Sakellariou, and E. R. Madeira, "Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm," in *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*. IEEE, 2010, pp. 27–34.

[125] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Computing Surveys (CSUR)*, vol. 31, no. 4, pp. 406–471, 1999.

[126] Y. Samadi, M. Zbakh, and C. Tadonki, "E-heft: Enhancement heterogeneous earliest finish time algorithm for task scheduling based on load balancing in cloud computing," *2018 International Conference on High Performance Computing and Simulation (HPCS)*, pp. 601–609, 2018.

[127] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities," in *High Performance Computing & Simulation, 2009. HPCS'09. International Conference on*. IEEE, 2009, pp. 1–11.

[128] R. Bajaj and D. P. Agrawal, "Improving scheduling of tasks in a heterogeneous environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 2, pp. 107–118, 2004.

[129] R. Armstrong, D. Hensgen, and T. Kidd, "The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions," in *Heterogeneous Computing Workshop, 1998.(HCW 98) Proceedings. 1998 Seventh*. IEEE, 1998, pp. 79–87.

[130] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima *et al.*, "Scheduling resources in multi-user, heterogeneous, computing environments with smartnet," in *Heterogeneous Computing Workshop, 1998.(HCW 98) Proceedings. 1998 Seventh*. IEEE, 1998, pp. 184–199.

[131] R. F. Freund and H. J. Siegel, "Guest editor's introduction: Heterogeneous processing," *Computer*, vol. 26, no. 6, pp. 13–17, 1993.

[132] C. Wu, X. Lin, D. Yu, W. Xu, and L. Li, "End-to-end delay minimization for scientific workflows in clouds under budget constraint," *IEEE Transactions on Cloud Computing*, no. 1, pp. 1–1, 2015.

[133] A.-M. Oprescu, T. Kielmann, and H. Leahu, "Budget estimation and control for bag-of-tasks scheduling in clouds," *Parallel Processing Letters*, vol. 21, no. 02, pp. 219–243, 2011.

[134] Z. Wu, Z. Ni, L. Gu, and X. Liu, "A revised discrete particle swarm optimization for cloud workflow scheduling," in *Computational Intelligence and Security (CIS), 2010 International Conference on*. IEEE, 2010, pp. 184–188.

[135] H. Zhao and R. Sakellariou, "An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm," in *European Conference on Parallel Processing*. Springer, 2003, pp. 189–194.

[136] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Characterization of scientific workflows," in *Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on*. IEEE, 2008, pp. 1–10.

[137] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy, "Task scheduling strategies for workflow-based applications in grids," in *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, vol. 2. IEEE, 2005, pp. 759–767.

[138] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[139] H. Jin, S. Ibrahim, L. Qi, H. Cao, S. Wu, and X. Shi, "The mapreduce programming model and implementations," *Cloud Computing: Principles and Paradigms*, pp. 373–390, 2011.

[140] Y. Samadi and M. Zbakh, "The impact of checkpointing interval selection on the scheduling performance of hadoop framework," *6th International Conference on Multimedia Computing and Systems (ICMCS)*, pp. 1–6, 2018.

[141] T. White, *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[142] D. Borthakur, "The hadoop distributed file system: Architecture and design," *Hadoop Project Website*, vol. 11, no. 2007, p. 21, 2007.

[143] H. Zhu and H. Chen, "Adaptive failure detection via heartbeat under hadoop," in *Services Computing Conference (APSCC), 2011 IEEE Asia-Pacific*. IEEE, 2011, pp. 231–238.

[144] R. K. Sahoo, M. S. Squillante, A. Sivasubramaniam, and Y. Zhang, "Failure data analysis of a large-scale heterogeneous server environment," in *Dependable Systems and Networks, 2004 International Conference on*. IEEE, 2004, pp. 772–781.

[145] Y. Li and Z. Lan, "Exploit failure prediction for adaptive fault-tolerance in cluster computing," in *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, vol. 1. IEEE, 2006, pp. 8–pp.

[146] F. Dinu and T. Ng, "Understanding the effects and implications of compute node related failures in hadoop," in *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*. ACM, 2012, pp. 187–198.

[147] H. Jin, K. Qiao, X.-H. Sun, and Y. Li, "Performance under failures of mapreduce applications," in *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*. IEEE, 2011, pp. 608–609.

[148] H. Wang, H. Chen, Z. Du, and F. Hu, "Betl: Mapreduce checkpoint tactics beneath the task level," *IEEE Transactions on Services Computing*, no. 1, pp. 1–1, 2016.

[149] R. A. Oldfield, S. Arunagiri, P. J. Teller, S. Seelam, M. R. Varela, R. Riesen, and P. C. Roth, "Modeling the impact of checkpoints on next-generation systems," in *Mass Storage Systems and Technologies, 2007. MSST 2007. 24th IEEE Conference on*. IEEE, 2007, pp. 30–46.

[150] Y. Samadi, M. Zbakh, and C. Tadonki, "Analyzing fault tolerance mechanism of hadoop mapreduce under different type of failures," *Cloud Computing Technologies and Applications (CloudTech 18) IEE*, 2018.

[151] S. Daneshyar and A. Patel, "Evaluation of data processing using mapreduce framework in cloud and standalone computing," *International Journal of Distributed and Parallel Systems*, vol. 3, no. 6, p. 51, 2012.

[152] J. Shafer, S. Rixner, and A. L. Cox, "The hadoop distributed filesystem: Balancing portability and performance," in *Performance Analysis of Systems & Software (ISPASS), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 122–133.

[153] Z. Yuan and J. Wang, "Research of scheduling strategy based on fault tolerance in hadoop platform," in *Geo-Informatics in Resource Management and Sustainable Ecosystem*. Springer, 2013, pp. 509–517.

[154] Y.-P. Kim, C.-H. Hong, and C. Yoo, "Performance impact of jobtracker failure in hadoop," *International Journal of Communication Systems*, vol. 28, no. 7, pp. 1265–1281, 2015.

[155] D. Sun, G. Chang, C. Miao, and X. Wang, "Analyzing, modeling and evaluating dynamic adaptive fault tolerance strategies in cloud computing environments," *The Journal of Supercomputing*, vol. 66, no. 1, pp. 193–228, 2013.

[156] M. Soualhia, F. Khomh, and S. Tahar, "Atlas: an adaptive failure-aware scheduler for hadoop," in *Computing and Communications Conference (IPCCC), 2015 IEEE 34th International Performance*. IEEE, 2015, pp. 1–8.

[157] S. Marzouk and M. Jmaiel, "A survey on software checkpointing and mobility techniques in distributed systems," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 11, pp. 1196–1212, 2011.

[158] A. Sangroya, D. Serrano, and S. Bouchenak, "Mrbs: A comprehensive mapreduce benchmark suite," *LIG, Grenoble, France, Research Report RR-LIG-024*, 2012.

[159] A. Sangroya, S. Bouchenak, and D. Serrano, "Experience with benchmarking dependability and performance of mapreduce systems," *Performance Evaluation*, vol. 101, pp. 1–19, 2016.

[160] Y. Chen, S. Alspaugh, and R. Katz, "Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 1802–1813, 2012.

[161] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: fair scheduling for distributed computing clusters," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. ACM, 2009, pp. 261–276.