Energy-efficient Workflow Scheduling with Budget and Deadline constraints in a Cloud Datacenter

Jean Etienne Ndamlabin Mboula

UNIVERSITÉ DE NGAOUNDERE THE UNIVERSITY OF NGAOUNDERE

Faculté des Sciences Faculty of Science

Order N°: _ _ _ _ _ _ _ _ _ _

DÉPARTEMENT DE MATHÉMATIQUES ET INFORMATIQUE
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

Unité de Formation Doctorale de Mathématiques, Informatique et Applications
Doctoral Training Unit of Mathematics, Computer Science and Applications

Energy-efficient Workflow Scheduling with Budget and Deadline constraints in a Cloud Datacenter

THESIS

Submitted in partial fulfillment of the requirements for the award of

Doctorat/Ph.D.

Speciality: System and Software in Distributed Environments

By

NDAMLABIN MBOULA Jean Etienne
Master of Science in Computer Engineering

Speciality: System and Software in Distributed Environments

Registration number: 02G034FS

Publicly defended on September 11, 2021

Jury:

KOLYANG DINA TAIWÉ	**Professor**	**The University of Maroua**	***President***
BITJOKA Laurent	**Professor**	**The University of Ngaoundere**	***Reporter***
NDIE DJOTIO Thomas	**Associate Professor**	**The University of Yaounde I**	***Reporter***
RISSET Tanguy	**Professor**	**INSA of Lyon, France**	***Examiner***
YENKÉ Blaise Omer	**Associate Professor**	**The University of Ngaoundere**	***Examiner***
TAYOU DJAMEGNI Clémentin	**Professor**	**The University of Dschang**	***Supervisor***
KAMLA Vivient Corneille	**Associate Professor**	**The University of Ngaoundere**	***Supervisor***

Year 2021

Energy-efficient Workflow Scheduling with Budget and Deadline constraints in a Cloud Datacenter

THESIS

Submitted in partial fulfillment of the requirements for the award of

Doctorat/Ph.D.

Speciality: System and Software in Distributed Environments

To

The University of Ngaoundere

Faculty of Science

Department of Mathematics and Computer Science
Doctoral Training Unit of Mathematics, Computer Science and Applications

By

NDAMLABIN MBOULA Jean Etienne

Master of Science in Computer Engineering

Speciality: System and Software in Distributed Environments

Registration number: 02G034FS

Supervisors :

TAYOU DJAMEGNI Clémentin	**Professor**	**The University of Dschang**
KAMLA Vivient Corneille	**Associate Professor**	**The University of Ngaoundere**

Year 2021

Author's declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: ..... DATE: .....

Dedication

I dedicate this work to

my parents: **MBOULA Fidèle** and **NGANTCHET Émérentine** (of blessed memories);

my lovely and exceptional wife, **BELA ENGOULOU J. Fanny**;
and my wonderful children: **NGANTCHET NDAMLABIN Melek Wisdom**
and **EYENGA NDAMLABIN Prokopé Gloire.**

Acknowledgements

First of all, my thanksgiving and acknowledgements are for the Almighty *GOD*, *JESUS CHRIST*, for his all-sufficient and amazing grace.

During this thesis, I benefited from the support of a lot of persons to whom I would like to express my deep gratitude.

I thank the Faculty of Science of the University of Ngaoundere, especially the Doctoral Training Unit of Mathematics, Computer Science and Applications.

I present here my gratitude to my thesis directors Pr TAYOU DJAMÉGNI Clémentin and Dr KAMLA Vivient Corneille. I appreciate the precious interventions, the advice and the kindness of Pr TAYOU, which boosted this thesis towards its successful completion. Dr KAMLA has been more than a director, I mean a father indeed. He provided support, advice, and encouragement to me from end to end of this thesis works.

May the members of the Jury find here the expression of all my gratitude for their participation. I especial thank Pr KOLYANG Dina Taïwé for presiding over the defence, and also Pr BITJOKA Laurent and Pr NDIE DJOTIO Thomas for agreeing to be my thesis' reviewers, and for their relevant comments on the manuscript.

I would like to thank Dr DAYANG Paul, the Head of the Department of Mathematics and Computer Science at the Faculty of Science of the University of Ngaoundere, and also Pr NTYAM Achille formerly Head of the same Department. I also thank Pr KAMGANG Jean-Claude and Pr YENKÉ Blaise Omer, respectively Head of the Department of Mathematics and Computer Science at the ENSAI, and Head of the Department of Computer Engineering at the UIT, University of Ngaoundere.

I would like to thank all the professors of mathematics and computer science of the University of Ngaoundere, for the lessons, advices and support that I benefited from them during my university studies. An especial acknowledgement to Dr TCHAKOUNTÉ Franklin for his comments and remarks that helped me to improve the quality of this document.

I am grateful to all the members of LAMEX laboratory of the University of Ngaoundere, among others, Dr FOTSA Jaurès, Dr TCHOUMI Stéphane, and in particular, Dr TCHAPPI Igor for his precious advice and support. I also appreciate my fruitful collaboration towards the end of this thesis with Dr Muhammad H. Hilman and his openness.

I cannot forget my parents in the Lord Apostle NJANKOUO Abdou Karios and his wife mama NJANKOUO Marie Lucie, the excellent Men of God and the beloved of the MIEG[1] Christian Ministry. In the same order, I thank Pst. & Ms. MENGALE Ernest & Shalom, Apostle & Ms. LOTAR André & Madeleine, and my wonderful beloved MANYO MANYO Eitel & Jacques, SAMDALLE Amaria, and ALMINE ORO KONDI.

Prior to conclude this acknowledgments part, I present all my unaccountable gratitude to my wonderful brothers and sisters (***MBOULA***) **Norbert**, **Rose**, **Claire**, **Louis-Laurent** and **Romuald**. May my mother-in-law, mama EYENGA Marie Bernadette, receive all my gratitude for her great heart and her undying encouragement and support. I also thank Ms. KAMLA Naderges, for her encouragement and multifaceted support.

May all those who I have not specifically mentioned the name, and who have shared my daily life during these years receive here the expression of all my gratitude.

[1]Mission Internationale d'Evangélisation de la Grâce

Contents

The enemy of the best is the good.
L'ennemi du meilleur c'est le bon.
(M. P. Karambiri)

There is no mountain anywhere, everybody's mountain is his ignorance.
Il n'y a de montagne nulle part, la montagne de chacun est son ignorance.
(D. O. Ayodepo)

Whatever is well conceived is clearly said, And the words to say it flow with ease.
Ce qui se conçoit bien s'énonce clairement, et les mots pour le dire arrivent aisément.
(Boileau)

List of Abbreviations

B

BDAS Budget Deadline Aware Scheduling

BoT Bag of tasks

C

CPU Central Processing Unit

CTTF Cost-time trade-off function

CTTWS Cost-Time Trade-off efficient Workflow Scheduling

CTTWS-W-IRITR Cost-Time Trade-off efficient Workflow Scheduling whitout IRITR

CTTWSDP Cost-Time Trade-off efficient Workflow Scheduling with Dynamic Provisioning

CTTWSDP-W-IRITR Cost-Time Trade-off efficient Workflow Scheduling with Dynamic Provisioning whitout IRITR

D

DAG Directed Acyclic Graph

DCCP Deadline Constrained Critical Path

DVFS Dynamic Voltage and Frequency Scaling

G

GRP-HEFT Greedy Resource Provisioning and modified HEFT

H

HEFT Heterogeneous Earliest Finish Time

I

IRITR Implicit Requested Instance Types Range

IT Information Technology

P

PPDPS Partition Problem based Dynamic Provisioning and Scheduling

S

SCTTEWS Structure-based Cost-Time trade-off and Energy-efficient Workflow Scheduling

SMWSH Structure-based Multi-objective Workflow Scheduling with Heterogeneous instance types

SMWSO Structure-based Multi-objective Workflow Scheduling with an Optimal instance type

V

VM Virtual Machine

VMs Virtual Machines

List of Figures

List of Tables

Abstract

Scientific workflows decompose complex scientific applications into smaller subsequent interdependent tasks that can be executed in serial or parallel. Their use has boosted scientific advancements in various fields such as biology, physics, medicine, and astronomy. However, scientific workflows are generally complex and have varied structures and characteristics that can have a significant impact on the result of a scheduling algorithm. Schedule a workflow consist to assign workflow tasks onto the resources of a computing infrastructure. Nowadays, the trend in information technology is the usage of Cloud computing environments to perform scientific workflow applications. However, cloud environments are experiencing a real problem of energy consumption. Inefficient resources management in cloud data centers has been identified as one of the main causes. That led to resources underutilisation, huge electricity bills, and reduction of the return of investment (ROI) for the cloud providers, and also high carbon dioxide emissions. As for the users, the respect of their defined deadline and budget is very important. In this thesis, we have proposed consecutively five workflow scheduling algorithms based on the structural properties of workflows. We have first investigated how to propose scheduling strategies to minimize both execution cost and execution time, which led to the proposal of two algorithms. Finally, we have investigated how to render our strategies more energy efficient. That led to the proposition of three scheduling algorithms aiming at minimizing the energy consumption, the execution cost, and the execution time. The three algorithms take advantage of the structural properties of the workflow as well as newly introduced scheduling concepts. At each step of our work, comparative simulations have been conducted between each of our proposals against state-of-the-art algorithms. Supported by adequate statistical tests, the analysis of the results reveals the levels of outperformance of our proposals both in the case of the two bi-objective algorithms than in the case of the three multi-objective ones aiming in addition at reducing energy consumption. The out-performance of the later ones in terms of energy-saving is established in 80% of types and workloads of workflows. Overall, one among the three, namely the Structure-based Multi-objective Workflow Scheduling with an Optimal instance type (SMWSO), is at least 50% more energy-saving, followed by our two other algorithms. As for the success rate, even though SMWSO scored overall the highest success rate, statistical tests revealed that there is no significant difference between our three algorithms and the baseline algorithm in terms of user satisfaction.

Keywords: *Cloud computing, Scientific workflows, Workflow scheduling, Budget, Deadline, Energy consumption minimization, Heuristics*

Résumé

Les workflows scientifiques décomposent des applications scientifiques complexes en plus petites tâches subséquentes et interdépendantes qui peuvent être exécutées en série ou en parallèle. Leur utilisation a boosté les progrès scientifiques dans divers domaines tels que la biologie, la physique, la médecine et l'astronomie. Cependant, les workflows scientifiques sont généralement complexes et ont des structures et des caractéristiques variées qui peuvent avoir un impact significatif sur le résultat d'un algorithme d'ordonnancement. Ordonnancer un workflow consiste à affecter ses tâches aux ressources d'une infrastructure informatique. De nos jours, la tendance dans les technologies de l'information est l'utilisation des environnements de cloud computing pour l'exécution des workflows scientifiques. Cependant, les environnements cloud connaissent un réel problème de consommation d'énergie. La gestion inefficace des ressources dans les centres de données cloud a été identifiée comme l'une des principales causes. Cela a entraîné une sous-utilisation des ressources, des factures d'électricité très élevées et une réduction du retour sur investissement pour les fournisseurs de cloud, ainsi que des émissions de dioxyde de carbone élevées. Quant aux utilisateurs, le respect de leur délai et budget lors de l'exécution est très important. Dans cette thèse, nous avons proposé consécutivement cinq algorithmes d'ordonnancement de workflow basés sur les propriétés structurelles des workflows. Nous avons d'abord étudié comment proposer des stratégies d'ordonnancement pour minimiser à la fois le coût d'exécution et le temps d'exécution, ce qui a conduit à la proposition de deux algorithmes. Enfin, nous avons étudié comment rendre nos stratégies plus écoénergétiques. Cela a conduit à la proposition de trois algorithms d'ordonnancement de workflow visant à minimiser la consommation d'énergie, le coût d'exécution et le temps d'exécution. Les trois algorithmes tirent parti des propriétés structurelles des workflows ainsi que de concepts d'ordonnancement nouvellement introduits. A chaque étape de nos travaux, des simulations comparatives ont été menées entre nos algorithmes et des algorithmes de pointe. Soutenue par des tests statistiques adéquats, l'analyse des résultats révèle les niveaux de surperformance de nos propositions aussi bien dans le cas des deux algorithmes bi-objectifs que dans le cas des trois algorithmes multi-objectifs visant en outre à réduire la consommation d'énergie. La surperformance de ces derniers en termes d'économie d'énergie est établie dans 80% des types et des charges de travail des workflows. Dans l'ensemble, l'un des trois, à savoir Structure-based Multi-objective Workflow Scheduling with an Optimal instance type (SMWSO), est au moins 50% plus économe en énergie, suivi de nos deux autres algorithmes. En ce qui concerne le taux de réussite, même si SMWSO a globalement obtenu le taux de réussite le plus élevé, les tests statistiques ont révélé qu'il n'y a pas de différence significative entre nos trois algorithmes et celui de la littérature en termes de satisfaction des utilisateurs.

Mots clés: *Cloud computing, Workflows scientifiques, Ordonnancement de workflow, Budget, Délai, Minimazation de la consommation en Energie, Heuristiques*

General Introduction

Cloud computing is the Information Technology (IT) paradigm that has successfully turned the vision of "computing utilities" into reality [7, 8]. As predicted by John McCarthy[2] in 1961: *"computer time-sharing technology might lead to a future in which computing power and even specific applications could be sold through the utility business model (like water or electricity)."* Afterwards in 1969 Leonard Kleinrock[3] stated that: *"As of now, computer networks are still in their infancy, but as they grow up and become sophisticated, we will probably see the spread of 'computer utilities' which, like present electric and telephone utilities, will service individual homes and offices across the country".*

Cloud computing aims at offering dynamic and non trivial pool of resources, exploitable on pay-per-use basis by users from their smartphones, tablets, laptops or personal computers, with an agreement to satisfy their requirements in terms of Quality of Service (QoS). Its adoption is ever increasing [9], due to its great flexibility, dynamicity and large variety of offers with competitive value for money. Nowadays even complex scientific applications (commonly known as HPC (High Performance Computing) or workflow applications) benefit from these assets of cloud environments. That trend led to the creation of large-scale data centers worldwide with thousands of servers by cloud providers. Consequently, there are increasing issues concerning the magnitude of energy consumption (resulting in huge costs of electricity bills) and contribution to the greenhouse effect due to CO_2 emissions [10, 11, 12].

Energy consumption, a real issue in the Cloud

The overall energy consumption of cloud computing was expected to be 1963 billion kWh by 2020, compared to 632 billion kWh in 2007 [11]. Only U.S. data centers consumed of 75 billion kWh of electricity annually which was equivalent to the output of around 26 medium-sized coalfired power plants. This energy usage is estimated to reach 140 billion kilowatt-hours annually, by 2020 [10].

That high energy consumption has been traced from several sources, among which the servers are the main power consumers [2, 13, 14]. In early 2010, researchers reported that under-utilization of cloud resources [2, 7, 15, 16] contributes to the high energy consumption in

[2]John McCarthy was an American computer scientist and cognitive scientist, co-founder of the discipline of artificial intelligence. https://en.wikipedia.org/wiki/John_McCarthy_(computer_scientist)

[3]Leonard Kleinrock is one of the chief scientists of the original Advanced Research Projects Agency Network (ARPANET) project which seeded the Internet.

cloud data centers, which is caused by the inefficient scheduling allocation of servers resources [2, 17]. From then, many works have been carried out to improve the energy-efficiency of cloud environments. From idle servers switching off [13], VMs/workload consolidation [18, 19, 20], to the Dynamic Voltage and Frequency Scaling (DFVS) [21, 22, 23, 24] techniques.

Recently in 2019, Flexera [25] reported that inefficient and time-consuming processes remain one issue to handle in order to maximize the ROI of both cloud providers and cloud users. In regard of that, significant energy saving still can be done through resource management techniques [17, 26] aiming at increasing the rate of resource usage. Virtual machines (VMs) consolidation is one of the well-known techniques, which aims at increasing the usage rate of cloud resources while saving energy consumption [27]. Consolidation of VMs involves migrations that can be expensive in terms of additional energy consumption, performance loss (hence cost), and this is not accounted for in some published models [26, 28, 29]. However, the reason of that avoidance of VMs consolidation by some researchers is the uncertainty nature of cloud environment which has been addressed by other researchers through the usage of learning algorithms and neurons networks [27, 30, 31].

Cloud as and opportunity for scientific workflow applications

The large acquisition or operational costs of dedicated infrastructure for scientific applications constitutes a barrier to many categories of users. A cheap, fast, and effective alternative was expected for long by such users. In fact apart greats laboratories, research groups and companies, small and medium scale enterprises, banks, and universities also need to exploit such environments to enhance their industrial processes, data analytic, business, and scientific discovery. Most computing service users prefer to pay only when they access computing services, rather than to run on their individual computers when the cost of ownership is high. Nowadays, Cloud environments are becoming a promising alternative of dedicated infrastructure for scientific applications, complex experiments based mainly on the analysis of large-scale data sets are gradually exploiting the assets of commercial clouds [32, 33, 34, 35, 36, 37, 38].

Almost all scientific areas are nowadays more complex and rely on the analysis of large scale data sets. It is therefore required to use an automated management process in a scalable way [39]. Scientific workflows have emerged as a suitable way of describing and structuring parallel computations and the analysis of large scale data sets [39]. Workflows are viewed by Scientists as key enablers for reproducibility of experiments involving large-scope computations [40]. Their successful use enhanced scientific advancements in various fields such as biology, physics, medicine, and astronomy [39, 40]. Scientific workflows have very complex structures

that can significantly impact the outcome of scheduling strategies [4, 5]. A workflow may contain hundreds or thousands of interdependent tasks [4] that are executed under a permanent dependency constraint where a task can only start its execution if the executions of its parents are completed. That means, allocated cloud resources can hardly be fully utilized as they might be inevitable unused gaps between tasks execution. This brings together the double challenge of complexities of clouds management along with that of the workflow structures. The level of difficulty of cloud workflow scheduling becomes very high, especially if we consider multiple user defined QoS parameters (e.g., deadline, budget). It is very important therefore to correctly determine the types and the number of cloud resources [41] to avoid the energy wastage and the violation of Service Level Agreement (SLA).

It has been advocated to design scheduling algorithms tailored for scientific workflows in order to take more advantage of clouds assets and deal with the complexity of both workflow application and cloud environments [4, 41].

Moreover, the Dynamic Voltage and Frequency Scaling (DVFS) has proved to be a simple and effective technique for the reduction of energy consumption [42, 24, 23, 22, 43, 21]. The structural properties of the workflows are generally employed when using DVFS technique. Some of the important parts of a workflow structure have been widely investigated, for instance the Critical Path (CP) (with the slack time reclamation/DVFS strategy), the distribution[4] tasks (with task duplication strategy), and the sequential and parallel tasks (with the tasks merging and slack time reclamation/DVFS strategies).

Research Problems and Objectives

As the structure of the workflow influence greatly the outcome of the scheduling strategy, several investigations have been done on strategy exploiting the structural properties of workflows. However, we found no work in the literature that investigates on the width of the workflow (the distribution of the number of tasks by level of workflow from entry to exit). We thing that it is important to look at it because of the precedence constraints existing in the workflow, when dealing with multi-objective scheduling. In fact, it is unlikely to use more VMs than the largest width of the workflow for its execution. Therefore, It is important to know or determine the optimal number and the types of VMs to use during the execution of a workflow, to avoid resource wastage and insure the respect of users as well as cloud providers requirements. In addition, little or almost no work combines several of the structural properties of workflows to see what it can achieve.

We advocate that homogeneity can produce better results if the suitable instance type

[4]tasks having more than one child, see Figure 4.1

is chosen, or a suitable (sub-)set of instance types are determined for the execution of the workflow. We further advocate that a suitable number of VMs if determined can help not only to produce better results in terms of user satisfaction, but also upgrade the maximization of VM utilization, and therefor the energy efficiency of the system. To the best of our knowledge, no work, or very few works determine suitable instance types set or an adequate number of VMs instances with an analytical approach. Some solutions use a *naive determination approach* in which it is at the end that one realize which types and the number of VMs have been used, leading in more cases to a wastage (too many provisioned VMs that are less utilized). Others are time consuming determination approaches, like greedy determination [44] and paths-based clustering determination [21, 45]. The paths-based clustering approach is better than the greedy one, however, its complexity and effectiveness are compromised if the workflow graph is strongly connected. Moreover, most of the solutions in the literature are effective only for a few types of workflow, while the types and structures of workflow are very complex and varied [4]. This is not conform with the recommendation [4, 41] of designing scheduling strategies that are effective no matter the type of workflow.

Research question

From the above-mentioned context and challenges, the research question of this thesis is the following:

How to build a workflow scheduling algorithm able to effectively assign the tasks of a workflow to cloud resources in order to minimize energy consumption, execution costs, and execution time, under user-defined budget and deadline no matter the type and the workload of the workflow?

Objectives

The main phases of a workflow scheduling algorithm are [3]:

1. *Resources provisioning*: it consists of selecting and provisioning the compute resources that will be used to run the workflow tasks.

2. *Scheduling* or *task allocation*: it consists of mapping each task onto the best-suited resource. Therefore, it can be divided into two stages:

   (a) *Task selection*: it consists of selecting a task among the non yet scheduled tasks of the workflow. It rely on a tasks prioritization.

(b) *task to VM mapping*: it consists of mapping the selected task onto the best-suited resource.

Our main objective in this thesis is therefore to: *propose scheduling strategies aiming at determining suitable order of tasks execution and suitable number and types of VMs for adequate task to VM mapping, in order to minimize energy consumption, execution costs, and execution time, in the respect of user-defined budget and deadline no matter the type and the workload of the workflow.*

To achieve this main objective, we have set ourselves the following specific objectives:

1. Study the structural properties of the workflows, and their influence over the cost, the makespan, and the energy consumption when workflow tasks are scheduled.
2. Design strategies to determinate suitable instance types (in the large range offered by loud providers), and adequate number of virtual machines to use in order to achieve the set (main) objective.
3. Design strategies based on workflow entry tasks, and workflow non-critical tasks along with DVFS to enhance the reduction of energy consumption, as well as of the execution cost and time.

Hypotheses

Our thesis works have been oriented by the following hypotheses:

1. Suitable Resources sub-set or Resources homogenization and suitable number of VMs can significantly improve scheduling outcome;
2. The distribution of the workflow widths and depths along with the user-defined budget and deadline are major keys for determining suitable instance types and number of VMs;
3. A fast execution of first tasks, using suitable types and number of VMs is a good track for effective scheduling.

Methodology

To respond to our preoccupation, we have proceeded as follow.

- We have first investigated how to propose scheduling strategies to minimize both execution cost and execution time.

- Afterwards, we have introduced the dynamicity in the scheduling process since cloud environments are dynamic in nature. In addition, we have investigated how to make our strategies effective no matter the type of the workflow to execute. This has been made through the analysis of the structural properties of workflows.
- Finally, we have investigated how to render our strategies more energy efficient. Since the energy is function the power and the time, the reduction of the execution time naturally contributes to energy reduction. However, there are ways to further reduce energy consumption without necessarily reducing the makespan. We have for instance the application of the DVFS technique on non-critical paths of the workflow.
- The evaluation of the performance of our proposed strategies has been made through comparative simulation against state-of-the-art algorithms.
- In each of the above steps we have conducted some studies on the structural properties of the workflows, and their influence over the cost, the makespan, and the energy consumption when workflow tasks are scheduled.

Contributions

The contributions of this thesis follow three main lines of strategies, and content five algorithms (see Figure 1) described as follows:

☛ Novelties:

1. *Line of strategies 1*: Proposition of a new technique for workflow scheduling in the cloud called the **Implicit Requested Instance Types Range (IRITR)**, combined with a trade-off function between execution time and cost. The IRITR aims at determining a range of VMs instance types that best suits the workflow execution, in order to avoid overbidding and underbidding that may lead to budget and deadline violation respectively. This line has three (#3) algorithms:
   - *(1^{srt} and 2^{nd}) algorithms* Two bi-objective heuristics for the minimization of execution cost and execution time based on the IRITR evaluation and the optimal number of VMs (Cost-Time Trade-off efficient Workflow Scheduling *(CTTWS)* and Cost-Time Trade-off efficient Workflow Scheduling with Dynamic provisioning *(CTTWSDP)*). The first one is static and naive in terms determination of number of VMs, while the second is dynamic, with structure inspired limitation of VMs. Their effectiveness has been proved through comparative simulation with state-of-the-art algorithms.

- *(3^{th}) algorithm* **S**tructure-based **C**ost-**T**ime trade-off and **E**nergy efficient **W**orkflow **S**cheduling *(SCTTEWS)*: extends the later one, and aiming at minimizing the energy consumption, the execution costs, and the execution time, under user-defined budget and deadline.

2. *Line of strategies 2*: Proposition of a new technique, which, based on structural properties of workflows and user-defined budget and deadline, determines an "optimal instance type" of VMs along with an "optimal number" of VMs for adequate provisioning. Here all the VMs used are of the same type ("optimal instance type"), and their number is limited by "optimal number". These new concepts have proved to be very effective in the achievement of the main goal of this thesis. This line has one (#1) algorithm:
   - *(4^{th}) algorithm* **S**tructure-based **M**ulti-objective **W**orkflow Scheduling with an **O**ptimal instance type *(SMWSO)*: it aims at minimizing the energy consumption, the execution costs, and the execution time, under user-defined budget and deadline. It is a real proof of concept, and scored better performance in most of our experiments than all the other proposed algorithms, and state-of-the-art algorithms.
3. *Line of strategies 3*: similar to the *Line 2*, but uses Heterogeneous instance types (exploit all the available types of VMs). This line has one (#1) algorithm:
   - *(5^{th}) algorithm* **S**tructure-based **M**ulti-objective **W**orkflow Scheduling with **H**eterogeneous instance types *(SMWSH)*: aiming at minimizing the energy consumption, the execution costs, and the execution time, under user-defined budget and deadline.

☛ Improvement of two scheduling techniques based on the structural properties of scientific workflows, namely *task duplication* and *pipelines merging and slacking*.

Thesis Organization

This thesis is structured as shown in Figure 2 and described as follows:

- A general introduction which sets the general framework for the thesis. Motivations and the context of research are explained while highlighting the research goal and objectives. It also presents the methodology followed to achieve the objectives and the main contributions of the work carried out;

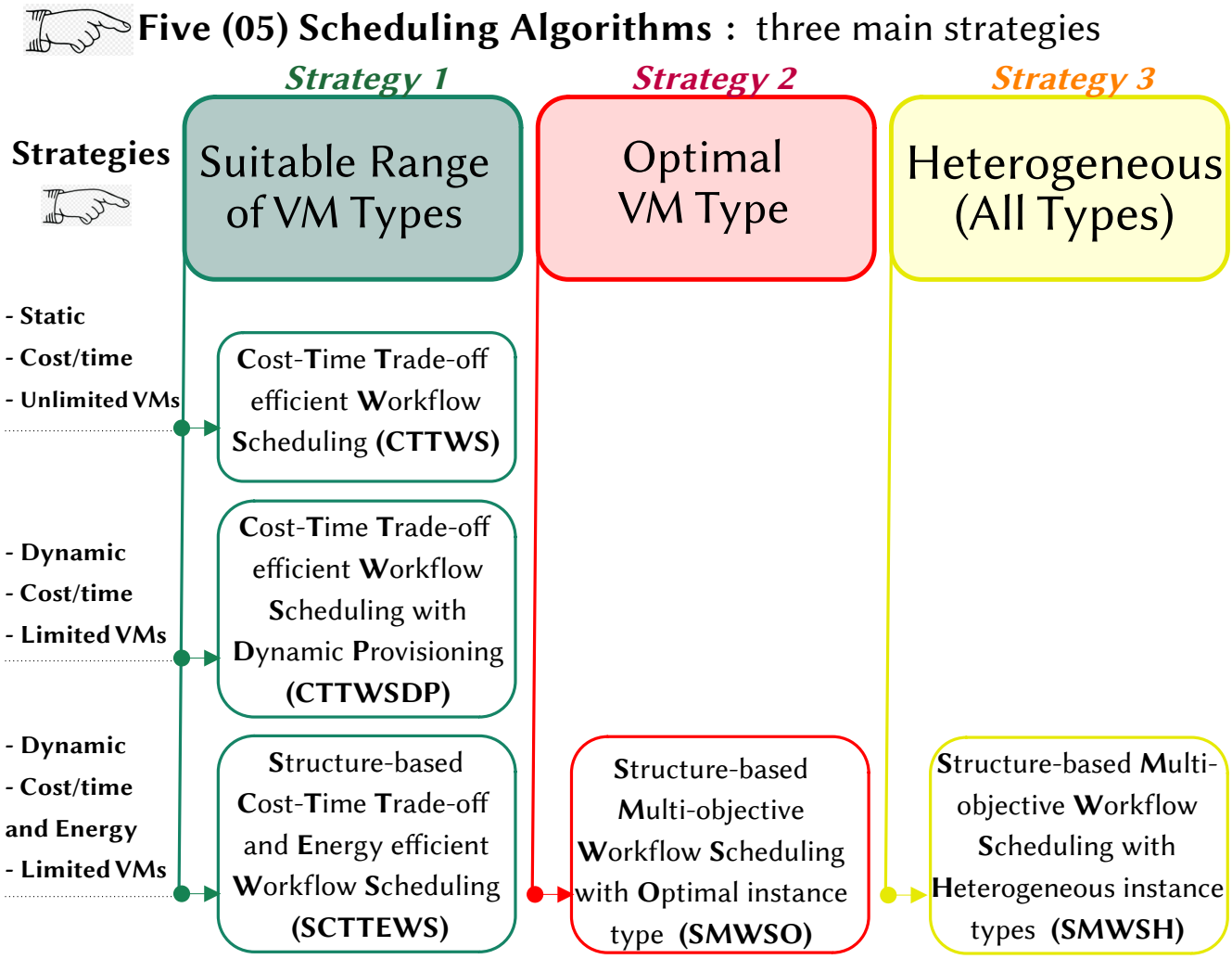


Figure 1: Thesis contributions

- Chapter 1 (Background) presents the cloud computing paradigm, scientific workflow management systems, workflow scheduling in cloud environments, and finally some examples of workflow scheduling algorithms;

- Chapter 2 (Cost-Time trade-off efficient workflow scheduling) presents a bi-objective cost/time scheduling approach based on two concepts, and named Cost-Time Trade-off efficient Workflow Scheduling (CTTWS). The first concept is a new technique aiming at determining a set of virtual machines (VMs) instance types for the execution of a given workflow so as to avoid overbidding and underbidding which could lead to a violation of the budget and the deadline respectively. The second concept is based on a cost/time trade-off function in order to select for each task of the workflow the VM instance suitable for achieving the targeted objective;

- Chapter 3 (Dynamic provisioning based workflow scheduling with budget and deadline awareness) presents a bi-objective cost/time scheduling approach with dynamic provisioning of VMs and limitation of resources (VMs) to the extent necessary to achieve the

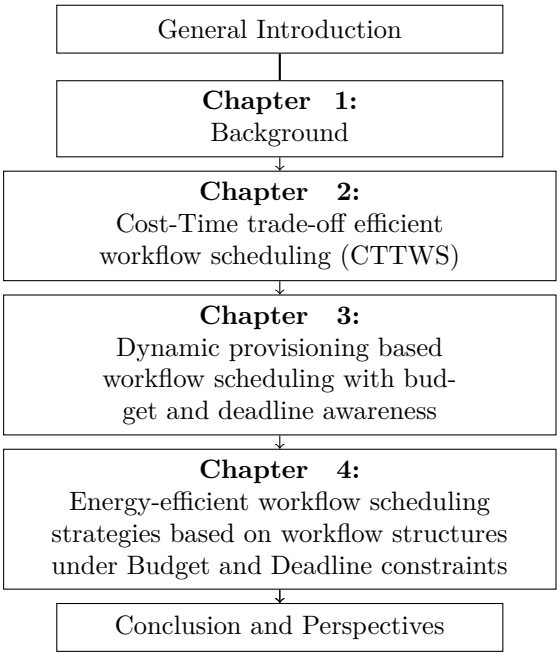


Figure 2: Thesis Organisation

objectives. The algorithm is named Cost-Time Trade-off efficient Workflow Scheduling with Dynamic provisioning (CTTWSDP) and extends CTTWS. CTTWSDP enhances the first concept used by CTTWS and adds a dynamic provisioning strategy of VMs. The improvement of the first concept has as objective to enhance the effectiveness of the algorithm no matter the type of the workflow and to maximize resource utilization.

- Chapter 4 (Energy-efficient workflow scheduling strategies based on workflow structures under Budget and Deadline constraints) firstly presents a study of the influence of some structural properties of workflows on scheduling in the cloud. Secondly, it proposes the construction of three multi-objective (minimization of cost, time, and energy consumption of VMs) heuristics based on the structural properties of workflows. The three algorithms take advantage of proposed techniques based on the structural properties of workflows to reduce execution cost, execution time as well as energy consumption. The first one, the Structure-based Cost-Time trade-off and Energy efficient Workflow Scheduling (SCTTEWS) algorithm is an improvement of the CTTWSDP which aims at reducing the energy consumption in addition to the execution cost and the execution time. The second one, the Structure-based Multi-objective Workflow Scheduling with an Optimal instance type (SMWSO) algorithm dynamically determines one suitable instance type among the available types of VMs, and only uses the VMs of that type. The third one, the Structure-based Multi-objective Workflow Scheduling with Heterogeneous instance types (SMWSH) is similar to the SMWSO algorithm but uses Heterogeneous instance types. The SMWSH algorithm has been designed purposely to highlight the strength of the SCTTEWS and

SMWSO algorithms. The three proposed algorithms use a limited number of VMs, based on the determination of an optimal number of VMs.

- The general conclusion (Conclusion and Perspectives) first recalls the context and the problem addressed in this thesis, then gives a summary of contributions provided within this thesis, and ends with some perspectives for future work.

CHAPTER

1 Background

Introduction

Cloud computing is the provision of IT resources on-demand via the Internet, with pay-as-you-go pricing. Instead of buying, owning, and managing physical servers and data centers, one can access to technology services such as computing power, storage, bandwidth, databases, and even end-users applications, on an as-needed basis, offered by cloud providers. If cloud computing is nowadays the most famous paradigm of the Information and Communication Technology industry (ICT), it has not been created from scratch. In fact, cloud computing is derived from the Cluster and Grid computing models, combined with utility and economic concepts [8, 46]. All those paradigms have strived to deliver *utility computing* vision [8], and it is the cloud computing paradigm that has successfully turn the vision into reality [7, 8, 47].

We can say that cloud computing has also become or is becoming a "high computing power utility", as researches on workflow management in cloud environments are intensively going forward to efficiently meet the constantly growing demands of complex scientific experiments that are based mainly on the analysis of large-scale data sets [32, 33, 34, 35, 37], which sometimes require high computing power [39]. Scientific workflow applications have emerged as the most suitable way of handling such complex scientific experiments [39]. Scientific advancements in various domains like biology, physics, medicine, and astronomy have been enhanced scientific through the successful use of scientific workflows [39, 40].

However, the attractive assets of cloud environments have not only been a source of gain, but also a trap. In fact, the ever-growing adoption of cloud encouraged providers to increase the underlying capacity of their data centers so that they can accommodate the increasing demand of new customers. Increasing the capacity and building large-scale data centers has caused a drastic growth in the energy consumption of cloud environments.

This energy consumption leads to the increase of the Total Cost of Ownership (TCO) of cloud providers, and of course, decreases the Return of Investment (ROI) of the cloud infrastructure. In addition to that, energy consumption has a significant impact on carbon dioxide (CO_2) emissions, which are estimated to be 2% of global emissions [48, 49]. That high energy consumption is traceable from several sources, among which the servers are the main power consumers [14, 13].

Therefore, paying attention to the energy consumption of cloud resources while managing uses' tasks is crucial, without ignoring the antagonist requirements of users in terms of quality of service (QoS), which generally are, to respect their budget and execution deadline.

This chapter presents the necessary concepts of cloud computing, workflows and workflow scheduling, for the general understanding of this report.

1.1 Cloud Computing Paradigm

1.1.1 What is cloud computing?

The type and generation of resources (hardware and software) are overgrowing with user demands; so linking a definition with today's ICT possibilities, as many attempts of cloud computing definition did, may lead to a continuous refinement of the definition. For this reason, there has recently been work on standardizing the definition of cloud computing, an example of which is the result of work by Vaquero and al. [50] who have compared more than 20 different definitions and have proposed a global definition. In the same direction, for the reason of standardization, we retain the definition proposed by the National Institute of Standards and Technology (NIST) [51].

Definition 1.1. *Cloud computing according to the NIST [51]. 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.*

Notwithstanding the approaches or technologies used to reach-out to its goal, that definition unveils the main goal of cloud computing.

Another definition that highlights the utility-oriented nature of cloud computing is given by Buyya et al. [8].

Definition 1.2. *Cloud computing according to Buyya et al. [8]. A cloud is a type of parallel and distributed system consisting of a collection of interconnected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements established through negotiation between the service provider and consumers.*

To reach these goals, several approaches or technologies are used. We can enumerate several characteristics of cloud computing (among which five essential), three service models, and four deployment models of the cloud computing technology [51].

Cloud computing characteristics

The five essential characteristics without which the cloud will no longer be the cloud are the following [51]:

1. *On-demand self-service:* A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.

2. *Broad network access:* Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).

3. *Elastic resource pooling:* The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, and network bandwidth.

4. *Rapid elasticity:* Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.

5. *Measured service:* Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.

Cloud Computing service models

The different models of services of the cloud are a good achievement of the "utility computing" objective. Utility computing describes a business model for on-demand delivery of whatsoever consumers need in return for the required remuneration from the supplier. Consumers pay providers based on usage ("pay-as-you-go"), similar to the way in which we currently obtain services from traditional public utility services such as water, electricity, gas, and telephony. In cloud computing, there are three main service models: Infrastructure-as-a-Service (IaaS),

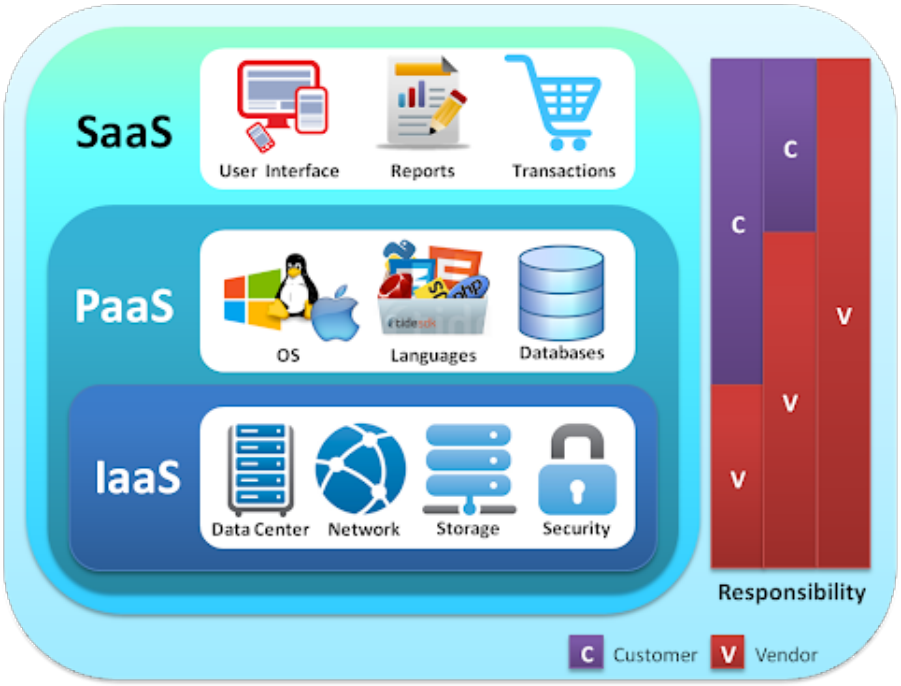


Figure 1.1: The cloud computing services models

Platform-as-a-Service (PaaS), and Software-as-a-service (SaaS). Each of these services models involves a certain level of responsibility for the customer and the supplier as presented in Figure 1.1.

- *Software-as-a-service (SaaS):* The SaaS is offering dedicated software and applications to cloud end users that are accessible on the Internet via a browser that are ready to be consumed.

  As examples we have: Gmail, Google docs and Facebook.

- *Platform-as-a-Service (PaaS):* The PaaS is providing the platform and the necessary Information technology (IT) environment for developers to implement and deploy their various services and applications on the Internet. The provider supplies and manages the underlying infrastructure.

  As examples we have: Google AppEngine, Microsoft Azure, IBM Cloud, Hadoop and Aneka.

- ***Infrastructure-as-a-Service (IaaS):*** IaaS is delivering virtualized resources called Virtual Machines (VMs) (storage, networking, servers, and other computing resources) to cloud customers. The IaaS is the principal service that benefits Workflow Management Systems (WfMSs).

  The topmost example of IaaS providers is Amazon Web Services (AWS) [1], which pro-

[1] https://aws.amazon.com

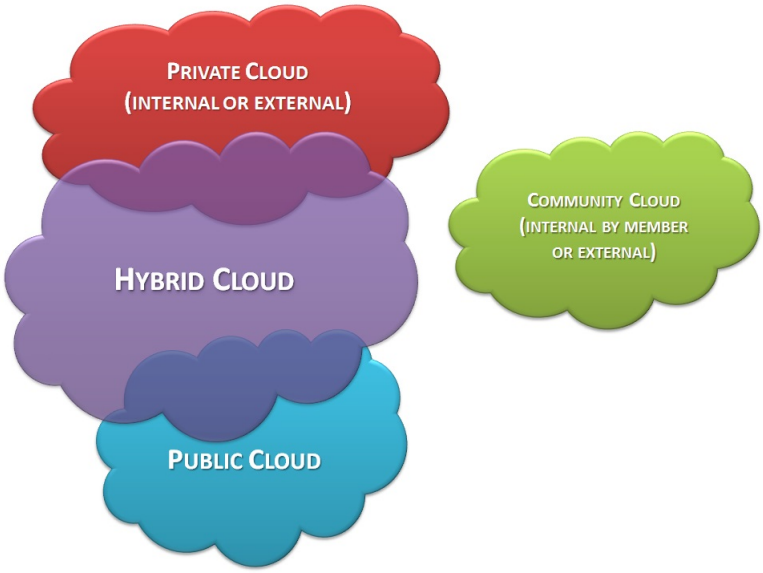


Figure 1.2: Types of clouds.

vides facilities for creating VMs, organizing them together into a cluster, and deploying applications and systems upon.

Types of clouds

It is possible to differentiate four types of clouds (see Figure 1.2), also known as Cloud Computing deployment models:

- ***Public cloud:*** The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider.
- *Private cloud:* The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises [2].
- *Hybrid or heterogeneous clouds:* The cloud is a combination of the two previous solutions and most likely identifies a private cloud that has been augmented with resources or services hosted in a public cloud.
- *Community cloud:* the cloud is characterized by a multi-administrative domain involving different deployment models (public, private, and hybrid), and it is specifically designed to address the needs of a specific industry.

[2] On-Premise defined: a solution hosted in-house and usually supported by a third-party. Off-Premise defined: a solution hosted by a third-party and usually supported by a different third-party.

1.1.2 Virtualization

The term virtualization is generally used to mean hardware virtualization, which plays a central role in cloud computing for the efficient delivery of Infrastructure-as-a-Service (IaaS) solutions as well as the other service models. Virtualization is a technology that hides physical hardware complexity and provides virtual resources for high-level applications. However, the concept of virtualization has not seen the day with the advent of cloud computing. In fact, virtualization technologies have a long trail in the history of computer science and have been available in many flavors by providing virtual environments at the operating system level, the programming language level, and the application level. Moreover, virtualization technologies provide a virtual environment for not only executing applications but also for storage, memory, and networking [1]. Its adoption has been spread like a bushfire in recent years with the development of the cloud more than it has been from its inception. Its renewed interest is due to the confluence of several phenomena [1]:

- *Increased performance and computing capacity:* Nowadays, the average end-user desktop PCis powerful enough to meet almost all the needs of everyday computing, with extra capacity thatis rarely used. Almost all these PCs have resources enough to host a virtual machine manager and execute a virtual machine with by far acceptable performance.

- *Underutilized hardware and software resources:* Hardware and software under-utilization is occurring due to (1) increased performance and computing capacity, and (2) the effect of limited or sporadic use of resources. Computers today are so powerful that in most cases only a fraction of their capacity is used by an application or the system.

- *Lack of space:* The continuous need for additional capacity, whether storage or compute power,makes data centers grow quickly. Companies such as Google and Microsoft expand their infrastructures by building data centers as large as football fields that are able to host thousands of nodes. This condition, along with hardware under-utilization, has led to the diffusion of a technique called server consolidation [3] for which virtualization technologies are fundamental.

- *Greening initiatives:* Recently, companies are increasingly looking for ways to reduce the amount of energy they consume and to reduce their carbon footprint. Data centers are one of the major power consumers; they contribute consistently to the impact that a company has on the environment. Maintaining a data center operation not only involves keeping servers on, but a great deal of energy is also consumed in keeping them cool.

[3]Server consolidation is a technique for aggregating multiple services and applications originally deployed on different servers on one physical server. Server consolidation allows us to reduce the power consumption of a data center and resolve hardware under-utilization

Infrastructures for cooling have a significant impact on the carbon footprint of a data center. Hence, reducing the number of servers through server consolidation will definitely reduce the impact of cooling and power consumption of a data center. Virtualization technologies can provide an efficient way of consolidating servers

- *Rise of administrative costs:* Power consumption and cooling costs have now become higher than the cost of IT equipment. Moreover, the increased demand for additional capacity, which translates into more servers in a data center, is also responsible for a significant increment in administrative costs. Computers–in particular, servers–do not operate all on their own, but they require care and feeding from system administrators.

These are the main causes of the diffusion of hardware virtualization solutions as well as the other kinds of virtualization. Historically [1], the first step towards the consistent adoption of virtualization technologies was made with the widespread of virtual machine-based programming languages: In 1995 Sun released Java, which has soon become the most famous and popular programming language among developers. The ability to integrate small Java applications called applets made Java a very successful platform, and with the beginning of the new millennium, Java played a significant role in the application server market segment, proving that the existing technology was ready to support the execution of managed code for enterprise-class applications. Followed by Microsoft in 2002 which proposed the .NET Framework, as an alternative to Java technology. In 2006, two of the three "official languages" used for development at Google, Java, and Python, were based on the virtual machine model. This trend of shifting toward virtualization from a programming language perspective demonstrated an important fact: The technology was ready to support virtualized solutions without a significant performance overhead. This paved the way to another and more radical form of virtualization that now has become a fundamental requisite for any data center management infrastructure.

Types of Virtualization and Characteristics of virtualized environments

Hardware virtualization in cloud environments There are many approaches adopted in the implementation of virtualization technology. Two important approaches are *Full Virtualization* and Paravirtualization. In the later one, VMs do not simulate the underlying hardware, and this uses a special API that a modified guest OS must use. Examples include Xen and VMWare ESX server. The full virtualization uses a special kind of software called a *hypervisor*. The hypervisor interacts directly with the physical server's hardware resources, such as the CPU and storage space, and acts as a platform for the virtual server's OSs. It helps to keep each virtual server completely independent and unaware of the other virtual servers running on the physical machine. Each guest server or the virtual machine (VM) is able to run its own OS (see Figures 1.3 and 1.4).

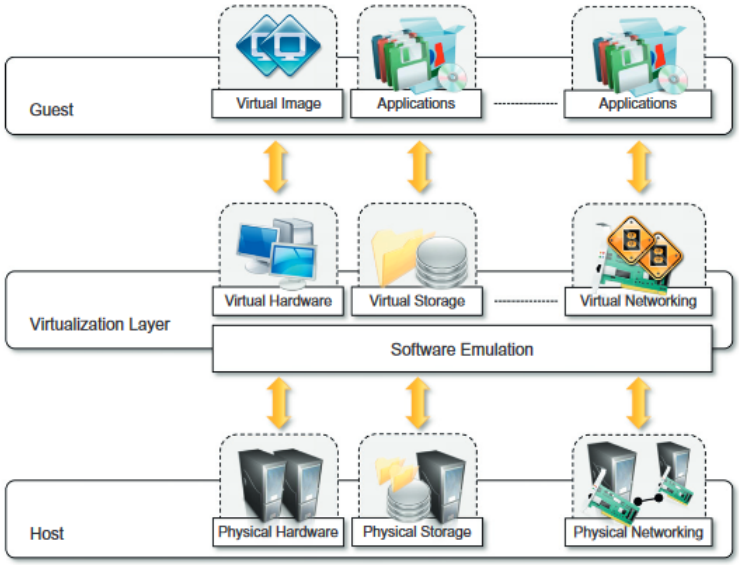


Figure 1.3: The virtualization reference model [1]

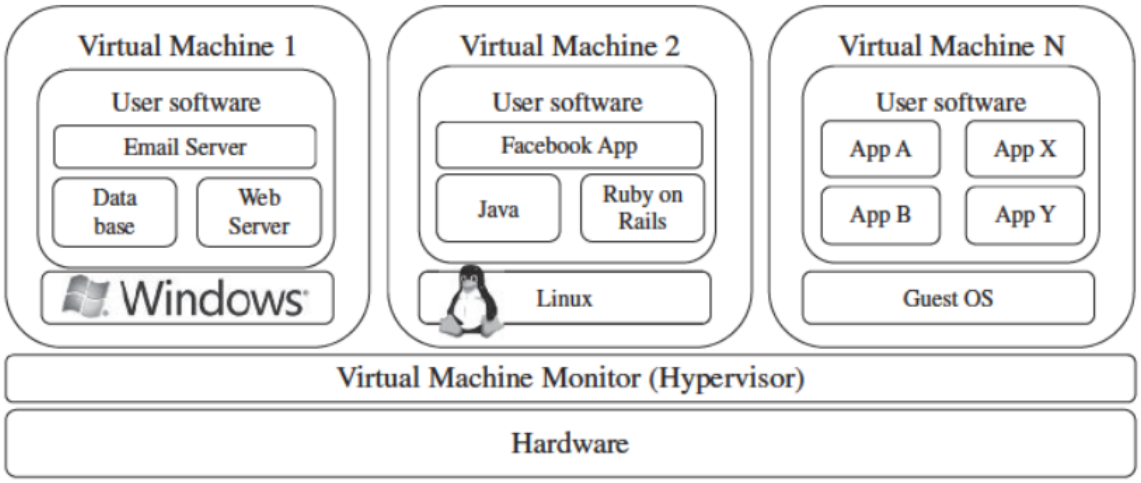


Figure 1.4: Hardware virtualization in cloud environments

A virtual hardware is used to provide compute power on demand offered as virtual machine instances [1]. The physical and virtual resources (CPU, memory) are mapped by the hypervisor between the virtual hardware and the hosts. The pricing model of virtual machine instances is generally defined in terms of dollars per hour, where the hourly cost is related to the characteristics of the virtual hardware.

Other types of virtualization in cloud environments Apart from hardware virtualization, there are also storage and network virtualization (see Figure 1.3).

- *Virtual storage:* As a physical machine has storage and memory, similarly virtual hard-

ware has, virtual storage and virtual memory. A virtual storage is delivered in the form of raw disk space or object storer complementing a virtual hardware offering that requires persistent storage. Virtual hardware is therefore a more high-level abstraction for storing entities rather than files.

- *Virtual network:* As physical machines communicate via a physical network, similarly virtual hardware also does so over a virtual network. This is the virtual networking, that identifies the collection of services that manage the networking among virtual instances and their connectivity to the Internet or private networks. Logical virtual networks are created from the underlying physical network. The physical networking components such as the router, switch, or network interface card could be virtualized by the hypervisor to create logical equivalent components [1, 52].

Characteristics of virtualized environments

Some of the main characteristics of virtualized environments are:

- *The increased security:* Virtualization allows an increased security. The ability to control the execution of a guest in acompletely transparent manner opens new possibilities for delivering a secure, controlled execution environment. All the operations of the guest are generally performed against the virtual machine, which then translates and applies them to the host. This level of indirection allows the virtual machine manager to *control* and *filter* the activity of the guest, thus preventing some harmful operations from being performed.

- *The isolation:* Virtualization allows providing guests–whether they are operating systems, applications, or other entities–with a completely separate environment in which they are executed, even though they are in reality executed in shared physical hosts.

- *The portability:* Generally, application depend open the presence of some tools and libraries for their execution. Virtualization allows the usage of a *virtual machine image* (see Figure 1.3) that can be plain operating systems or can have software installed on them, such as databases, application servers, or other applications needed for the execution of users application. This grant a portability property to the virtualization technology.

1.1.3 Quality of Service (QoS)

The Quality of Service (QoS) in a cloud computing environments is expressed with more or less high-level parameters. Compliance with these levels of values for the various parameters is the

subject of a contract between a cloud user and its cloud service provider and is called a Service Level Agreement (SLA).

These different QoS parameters within a cloud can be classified into four categories [53]: Performance, Dependability, Security & Data, and Cost. Among the numerous existing parameters, we will just present those considered in the evaluation of our work, an exhaustive list can be found in [53].

- *Performance category:*

  Definition 1.3. *Execution Time. The Execution time depends on the complexity of the request to be executed (in number of instructions) and on the capacity of the virtual machine in which it is processed.*

  Definition 1.4. *Response Time. The response time is the time elapsed between sending a user request and when it receives the response from the service. It is the time required to make a service available to a user and can serve to measure the efficiency of a service.*

- *Dependability category:*

  Definition 1.5. *Reliability according to Endo et al. [54]. The Reliability can be defined as the ability of an item to perform its required functions for a stated time and under operational conditions.*

  Definition 1.6. *Reliability according to Zhang and Chakrabarty [55]. The Reliability of the system is the probability of execution of task without any failure.*

  Additionally, reliability of computing node is of prime concern specially for computation intensive application [21].

- *Cost category:*

  Definition 1.7. *Service Cost. The service cost is defined by the cloud provider relatively to the type of service purchased by the user and the duration of the service. In the case of the leasing of virtual machines, the common billing model is hourly-based. That is, for each partial hour consumed will be rounded up to a full hour, such that 1 hour and 1 minute (61 min.) will be considered as 2 hours (120 min.) of utilization. The service cost will then be obtained by multiplying the total number of hours per the unit hour cost of the VM.*

  Definition 1.8. *Energy Cost or Energy consumption. The energy cost of an equipment, expressed in kilowatt-hour (kWh), represents the energy necessary to operate it over a given period of time. It is a function of the power (in Watt) delivered by this equipment (physical machine for example) and the duration of use (in Hours).*

The kilowatt-hour (kWh) is a composite unit of energy equal to one kilowatt (kW) of power sustained for one hour.

1.1.4 Issue of High Energy Consumption in the Cloud

As stated in the introductory part of this chapter, energy consumption is a central issue in cloud computing environments. As described by Moore's law [56], the increasing of the capacity of data centers has been possible through the efficient system design and increasing density of the component. This has continuously increased the performance per watt ratio, yet the total power consumed by computer systems has hardly decreased. For example, in 2006 the cost of power consumption by data centers in the United States was 4.5 billion US dollar, and doubled in 2011 [19, 57]. It is therefore crucial to know the main reasons for that issue power/energy consumption in the cloud.

Sources of high energy consumption

There is no doubt that energy consumption also depends on cooling equipment and power delivery infrastructure (since they are continuously supplying power to equipment). however, half of the data centers energy is wasted mostly due to the inefficient allocation of servers resources [2, 17]. Efficiency handling can be done at various level of a computing systems [2] (see Figure 1.5).

Though it is difficult to have precise information about the resources usage of Cloud providers, many researches give the tendency of cloud resources utilization. There is a real under-utilization issue of cloud infrastructures [58, 15, 16]. About 52% of cloud resources are currently highly underutilized [58], many been hardly use or totally sitting unused. Researchers on Google traces reveal that the cluster is only 20% – 40% utilized [15].

In regard of the large under-utilization of resources in data centers, significant energy saving can be done through resource management techniques [15, 26] aiming at increasing the rate of resource usage. VMs consolidation is one of the largely proposed technique that increasing the rate while saving energy consumption.

Energy consumption reduction approaches

As it can be inferred from Figure 1.5, there is three main levels from which one can reduce the energy consumption of the system (physical machine level, *virtual machine level*, application level) through efficient management. While the application level is of the responsibility of the PaaS user, the two others have to be handle by the provider.

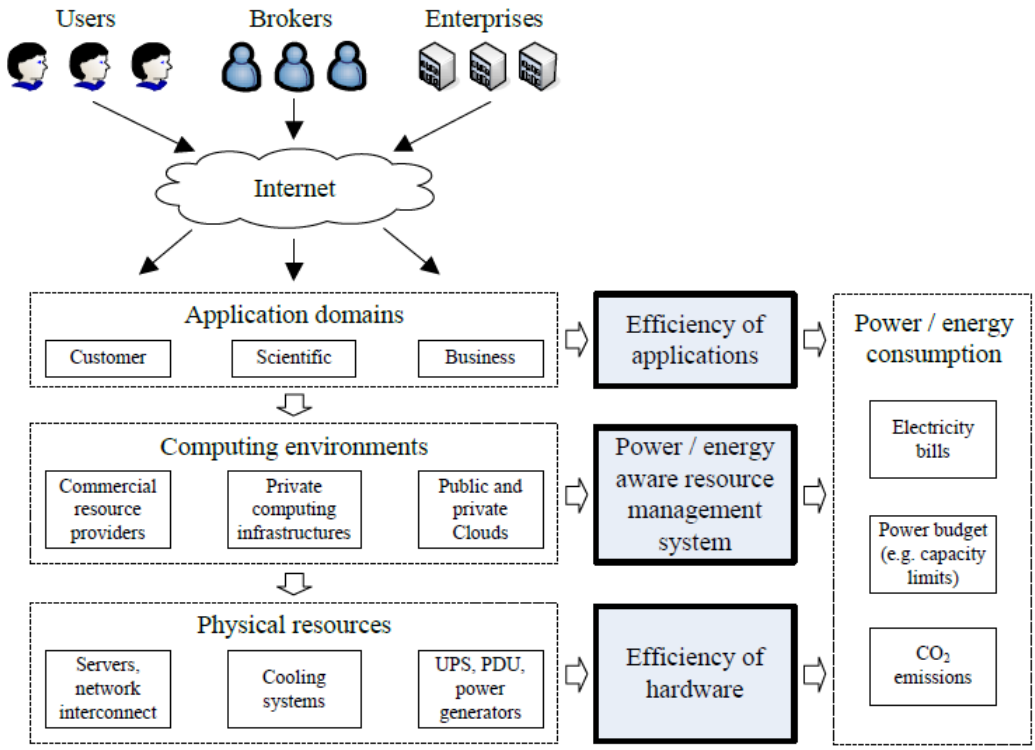


Figure 1.5: Energy consumption incurred at divers levels in computing systems [2]

Among the existing energy reduction approaches we have the following:

- *Switching idle servers off:* this technique consists to shut down servers when there are not in use. It can significantly reduce server consumption, as it ensures near-zero energy consumption by turning servers off. However, previous works that took this approach had difficulties to assure service-level agreement due to the lack of a reliable tool for predicting future demand to assist the turning off/on decision-making process [13].

- *VMs/workload consolidation:* In cloud environments, the ability to migrate VMs at run-time from one physical host to another enables the technique of energy-efficient dynamic VM consolidation. That technique increases the workload of some host at the expense of others, which can then be turned off using the first technique. That technique has been extensively investigated by Beloglazov et *al.* [27, 57, 20].

- ***The Dynamic Voltage and Frequency Scaling (DVFS)****:* [59] this technique consists of dynamically changing the frequency of the CPUs of physical machines according to their usage rates. The goal is to decrease the supply voltage (thus the clock frequency) of the CPU in order to consume less power.

  A reduction in frequency and voltage inevitably results in a decreasing of the CPU power due to the nature of today's CMOS (Complementary Metal Oxide Semiconductor) circuits [60, 61]. This can be illustrated by Figure 1.6. In such a circuit, the dissipated power

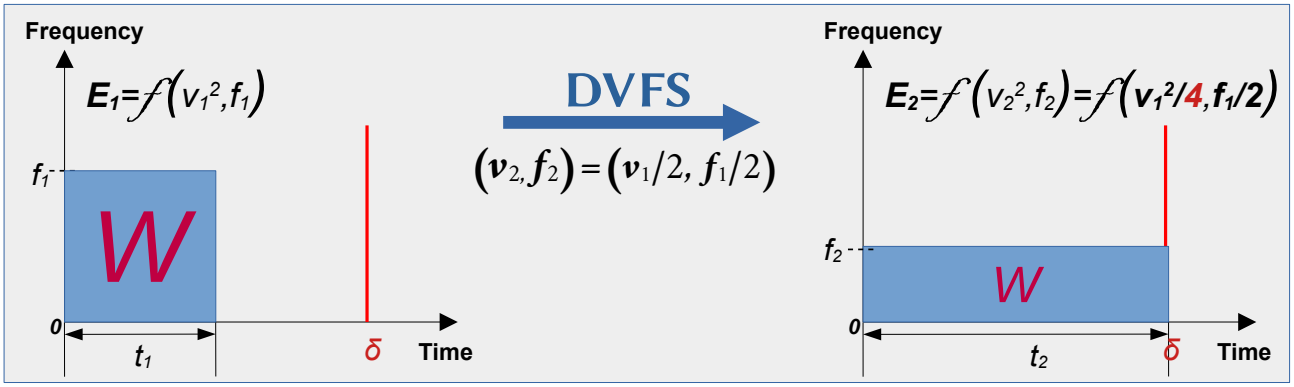


Figure 1.6: Illustration of the energy consumption reduction via the DVFS technique.

of a CPU is made up of two parts [62]: static and dynamic; the dynamic part been the dominant one and estimated according to equation (1.1) [6, 62, 63, 64].

$$P_{dynamic} = a \times C \times V^2 \times f \tag{1.1}$$

where a is the number of switches per clock cycle, C the effective charged capacitance, V the supply voltage and f its corresponding frequency. We then have V and f belonging to the ranges $[V^{min}, V^{max}]$ and $[f^{min}, f^{max}]$ respectively, where V^{min} and f^{min} are the values in the idle state of the CPU.

According to equation (1.1), a linear reduction in voltage V implies a quadratic power reduction provided by the component. However, a decrease in the voltage V also implies a reduction in the switching speed of the transistors making up the CMOS, which inevitably leads to a decrease in the theoretically possible maximum frequency of the processor. CPU idle and operational states are known as *C-states* and *P-states* [6], and the switching time between the P-states couple frequency/voltage (see Table 1.1) is very fast and around ten milliseconds [65].

Most modern CPUs including mobile, desktop, and server systems support DVFS.

Table 1.1: P-States for the Intel Pentium M Processor at 1.6GHz [6]

P-State	Frequency	Voltage	Estimated power
P0	1.6 GHz	1.484 V	~ 25 Watts
P1	1.4 GHz	1.420 V	~ 17 Watts
P2	1.2 GHz	1.276 V	~ 13 Watts
P3	1.0 GHz	1.164 V	~ 10 Watts
P4	800 MHz	1.036 V	~ 8 Watts
P5	600 MHz	0.956 V	~ 6 Watts

1.2 Workflows Management System (WfMS)

The general concept of workflow had a long journey in the business world before the advent of scientific workflow. A whole industry of tools and technologies dedicated to workflow management has been developed and marketed to meet the needs of commercial enterprises. The Workflow Management Coalition [4] (WfMC), founded in August 1993, is a non-profit, international organization of workflow vendors, users, analysts, and university/research groups, has developed a large set of reference models, documents, and standards.

1.2.1 Concepts Definition

Independently to the domaine, WfMC defines the concept of workflow as follows.

Definition 1.9. *Workflow according to the WfMC (1996). Workflow is the automation of business process, in whole or part during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.*

The particularity of scientific workflows is that they tend to change more frequently and may involve very voluminous data translations. Almost all scientific areas are nowadays more complex and rely on the analysis of large scale data sets; it is therefore required to use an automated management process in a scalable way [39]. Scientific workflows decompose complex scientific applications into smaller subsequent interdependent tasks that can be executed in serial or parallel according to its structure. Scientific advancements in various fields such as biology, physics, medicine, and astronomy [39, 40] have been boosted via successful use of scientific workflows. Scientific workflows are generally modelled as a Directed Acyclic Graph (DAG).

1.2.2 Workflows Management System architecture

The notion of Workflows Management System (WfMS) in grid and in cloud environments is largely described by the authors of [3, 66, 67, 68, 69].

The execution of workflows in cloud environments is done via a Cloud Workflow Management System (CWfMS). Figure 1.7 presents a reference architecture common to most CWfMS implementations, meanwhile, some specificity or simplification can be done from one system to another [3].

[4] https://www.wfmc.org/

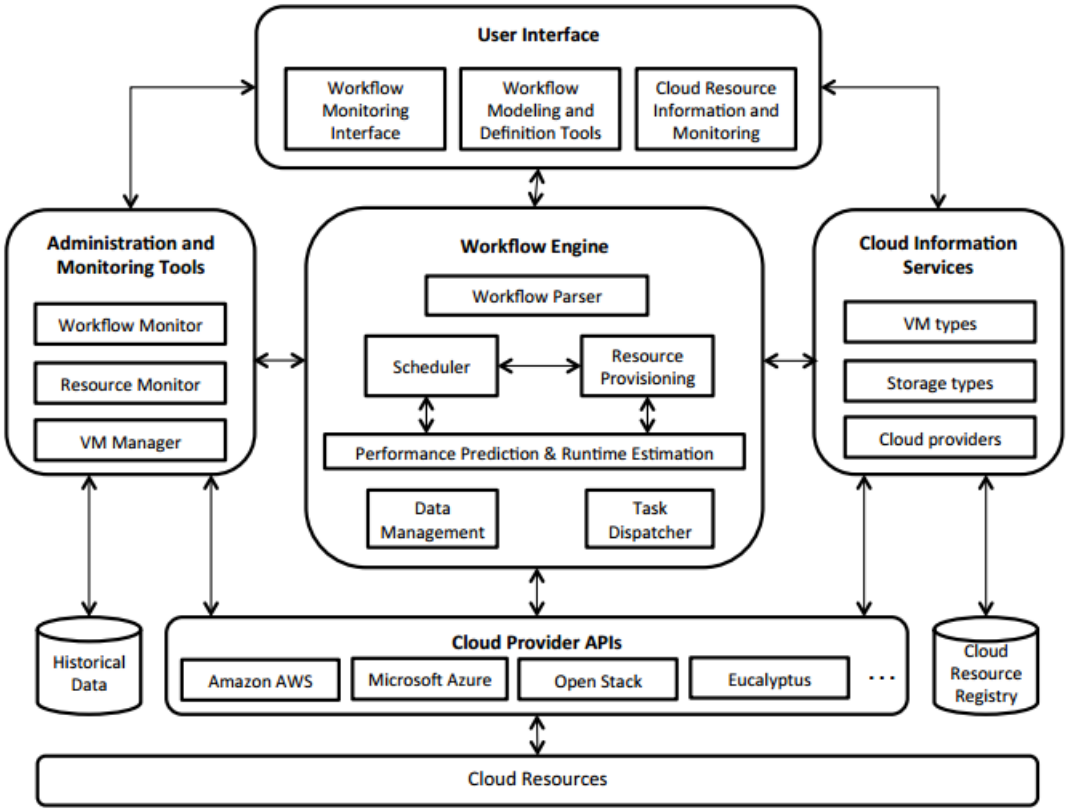


Figure 1.7: Reference architecture of a Workflow Management System [3]

- *User Interface:* via the user interface, users can create, edit, submit, and monitor their applications.

- *Workflow Engine:* The workflow engine is the core of the system. Its responsibility is to manage the whole execution process of the workflow. The workflow is first parsed (according to its format, eg. XML) by the workflow parser, which afterward creates the workflow tasks according to their interdependencies. The scheduler works along with the resource provisioning modules to plan the execution of the workflow tasks. The resource provisioning module's aim is to select and provision the cloud resources, allowing to the scheduling component to apply mapping policies between tasks and available resources, both processes based on the QoS requirements and scheduling objectives.

- *Administration and Monitoring tools:* The administration and monitoring tool is composed of modules working together to enable dynamic and permanent surveillance of workflow tasks, as well as the management and the performance of leased VMs.

- *Cloud Information Services (CIS):* the CIS provides to the workflow engine information about different cloud providers, the resources they offer including their characteristics and prices, location, and any other information required by the engine to make the resource

selection and mapping decisions.

- *Cloud Provider APIs:* These APIs enable the integration of applications with cloud services, as for instance the on-demand provisioning/de-provisioning of VMs, the monitoring of resource usage within a specific VM, access to storage services to save and retrieve data, transferring data in or out of their facilities, and configuring security and network settings, among others.

The main challenge in workflow management is the scheduling of workflow tasks [3, 66, 39] which is a well-known NP-complete problem [70].

1.2.3 Real world examples of scientific workflows

Here we present five well-known real-world workflows from different scientific areas. There are: Montage workflow, CyberShake workflow, Epigenomics workflow, Inspiral/Ligo workflow, and Sipht workflow [4].

(a) **Montage workflow:** the Montage workflow is generated based on the concept of astronomy. The Montage application from the astronomy field is used to generate custom mosaics of the sky based on a set of input images. The workflow calculates the geometry of the output mosaic on the sky and re-projects the flux in the input images. Finally, a background radiation model is generated based on the input images to achieve common flux scales and background levels across the mosaic. The normalized images are used to generate the final mosaic. Most of its tasks are characterized by being I/O intensive while not requiring much CPU processing capacity.

(b) **CyberShake workflow:** the CyberShake workflow is used by the Southern California Earthquake Center (SCEC)[5] to characterize earthquake hazards by generating synthetic seismograms. CyberShake relies on scientific workflows to provide the reliability, robustness, and automation needed to reach the necessary computational scale. CyberShake can be classified as a data-intensive workflow with large memory and CPU requirements.

(c) **Epigenomics workflow:** the Epigenomics workflow was developed by the University of Southern California (USC) Epigenome Center[6] and the Pegasus Team[7]. The Epige-

[5]The SCEC is a research center which studies why and how earthquakes occur, evaluate their effects, and help societies prepare to survive and recover. https://www.scec.org/

[6]The USC Epigenome Center conducts genome-scale epigenetic and genetic data production and analysis, technology development, and cutting-edge epigenomic and population-based genomic research. http://epigenome.usc.edu.

[7]The Pegasus project encompasses a set of technologies that help workflow-based applications execute in a number of different environments including desktops, campus clusters, grids, and clouds. Pegasus bridges the scientific domain and the execution environment by automatically mapping high-level workflow descriptions onto distributed resources. https://pegasus.isi.edu/

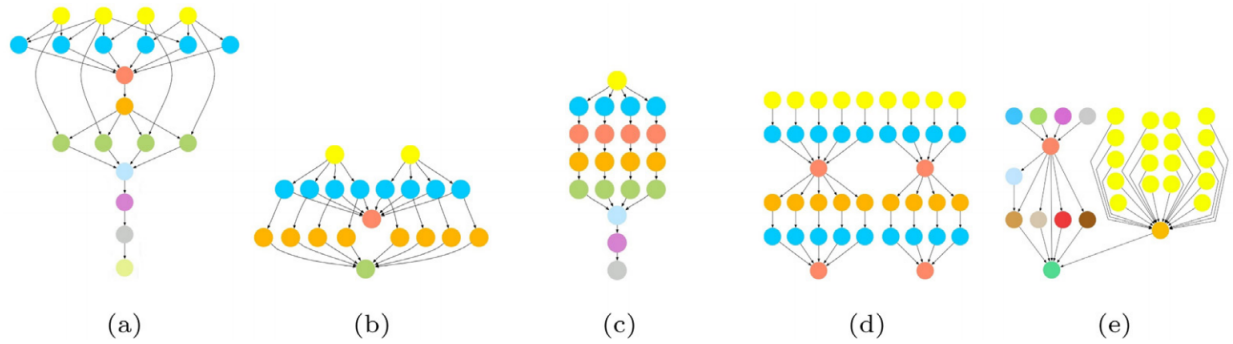


Figure 1.8: Scientific workflows structure sample used in our experiments [4] : (a) Montage, (b) CyberShake, (c) Epigenomics, (d) Inspiral and (e) Sipht.

nomics workflow is generated based on the concept of bioinformatics which automates the execution of various genome-sequence operations. Most of the tasks of the workflow require high CPU utilization and low I/O utilization.

(d) **Inspiral/Ligo workflow:** The Laser Interferometer Gravitational-Wave Observatory (LIGO) workflow is used to detect network of gravitational-wave with observatories in Livingston and Hanford [71]. LIGO workflow is used to detect gravitational waves in physics. LIGO is composed mostly of CPU intensive tasks with high memory requirements.

(e) **Sipht workflow:** SIPHT means sRNA Identification Protocol using High-throughput Technology. SIPHIT workflow is generated by the researcher of Harvard University based on small untranslated RNAs (sRNAs) that control several processes of a bacteria such as secretion or virulence [72]. The main purpose of such type of workflow is to endocde the genes of all bacterial replicas, stored in National Center for Biotechnology Information (NCBI) database[8]. Most of SIPHT tasks have low CPU and high memory utilization.

Samples structures of these workflows can be seen in Figure 1.8, their complete description and characterization are given by Juve et al. [4].

It appears that workflows structures are very varied and complex. Therefore, a scheduling algorithm should not ignore or consider only one particular workflow structure but adapt to the different possible structures [4, 41].

1.3 Workflow Scheduling in Cloud

[8]The National Center for Biotechnology Information advances science and health by providing access to biomedical and genomic information. https://www.ncbi.nlm.nih.gov/

1.3.1 Workflow scheduling taxonomy

We briefly present here some taxonomies of workflow scheduling, important for the understanding of this thesis; exhaustive description are proposed in [3, 66]. In Figures 1.9 and 1.10, the notions in bold are those characterizing the different contributions in this thesis.

Application model taxonomy

This taxonomy describes the ability of an algorithm to schedule either a single or multiple workflows, that is the multiplicity of workflow handled (see Figure 1.9). Single workflow algorithms are designed to optimize the schedule of a single workflow. Multiple workflows and workflow ensembles are similar categories and correspond to the scheduling of many scientific workflows. The difference is that in workflow ensembles, the workflow instances are interrelated because their combined execution produces a desired output.

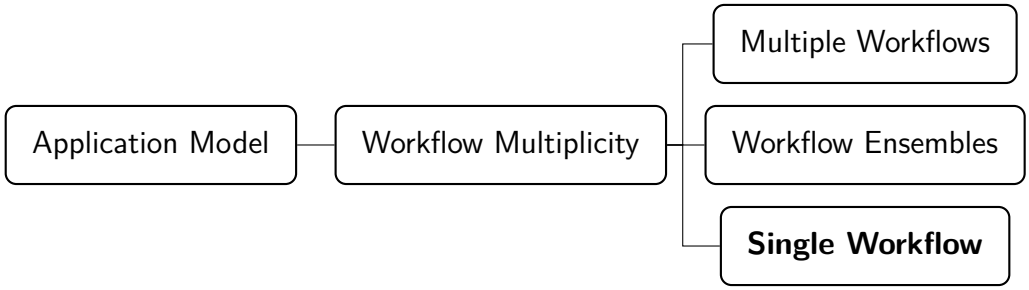


Figure 1.9: Application model taxonomy

Scheduling model taxonomy

In this section, we identify some characteristics of workflow scheduling that are relevant for our work (see Figure 1.10). We have the Task-VM mapping dynamicity, the resource provisioning strategy, the scheduling objectives, and optimisation strategy [3, 66].

Task-VM mapping dynamicity: workflow scheduling algorithms can be classified as either static or dynamic. Static ones are algorithms in which the task to VM mapping is produced in advance and executed once. The tasks of the workflow are assigned in the pre-defined set of VM instances based current state information about the resources in the servers. And possible variation of the performance of the computing resources in the servers are not considered. At the contrary, the dynamic algorithms make task to VM assignment decisions at runtime based on the current state of the system and the workflow execution. In addition to these two classes, we identify a third hybrid one, in which algorithms combine both approaches [3].

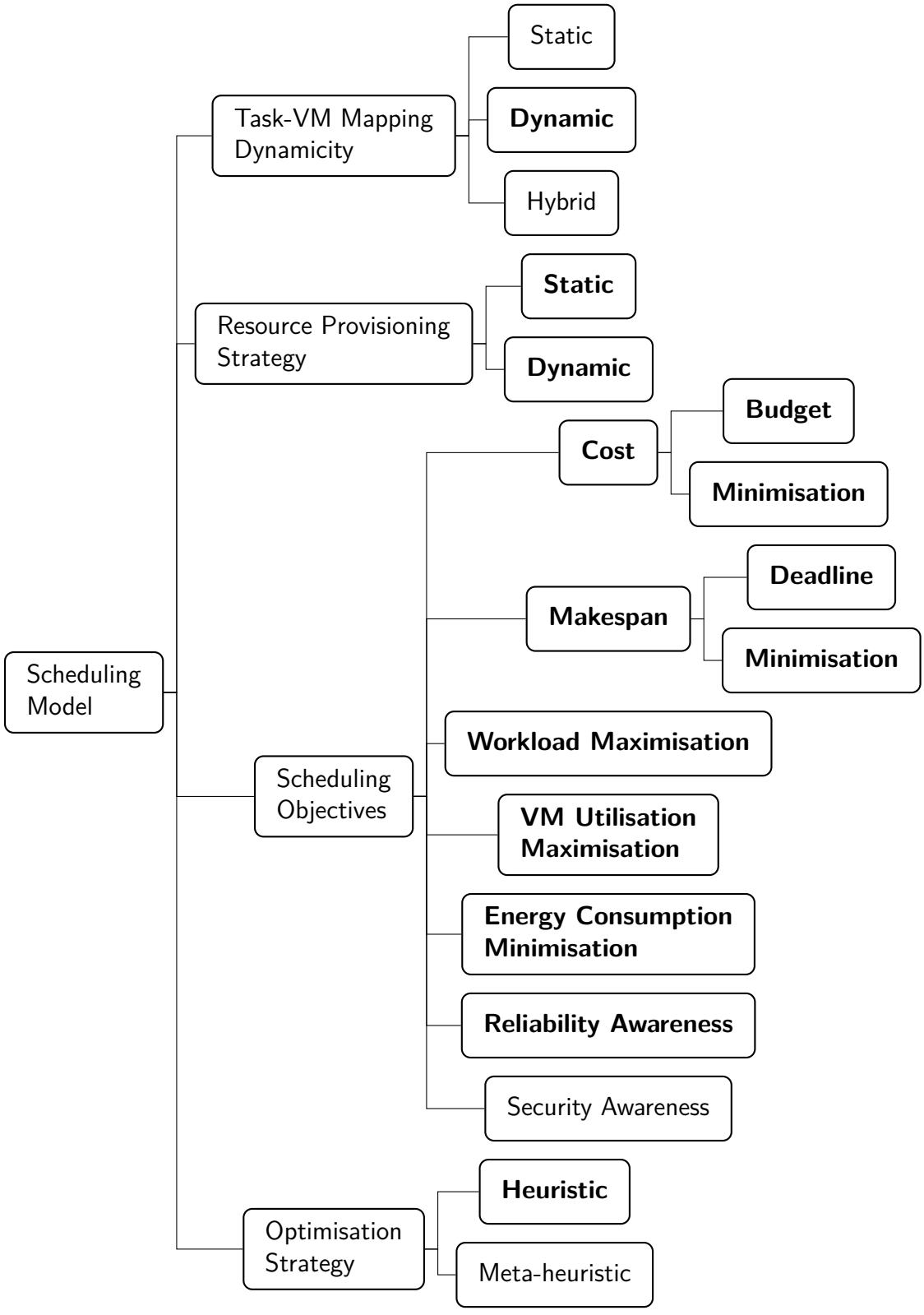


Figure 1.10: Scheduling model taxonomy

Resource provisioning strategy: Similarly to the task to VM mapping, algorithms may adopt a static or dynamic resource provisioning approach. In the static resource provisioning, the VM pool configuration is considered ready before the execution of the workflow. In the

dynamic provisioning, on the other hand, the VM pool configuration is made or refine at runtime, selecting which VMs to create, which VMs to keep active, which ones to lease, and which ones to release as the workflow execution progresses.

Scheduling objectives: examples of scheduling objectives are:

1. the minimization of the execution cost or the respect of user-defined budget: for the user, reducing execution cost is of a capital importance. User sometimes have a cap on the amount of money spent on resources (i.e. budget). Minimizing execution cost and taking into account user-defined budget in a scheduling strategy is crucial.

2. the minimization of the execution time or the respect of user-defined deadline: as in the case of the execution cost, execution time and deadline are of a capital importance for cloud users. Therefore, scheduling strategies must strive to reducing the makespan while mapping workflow tasks on cloud resources and be aware of user-defined deadline.

3. the maximization of the workload aiming at maximizing the amount of work done, that is, the number of workflows executed;

4. the maximization of VM utilisation: idle time slots in provisioned VMs are deemed as a waste of money as they were paid for but not utilised and as a result, algorithms try to avoid them in their schedules. Minimising idle time slots and maximising the utilisation of resources helps reducing execution cost as well as energy consumption, engraving profit to providers.

5. the minimization of energy consumption: The current industries, organizations, and governments are really concern about the reduction of energy and to reduce the carbon footprint. For cloud computing providers, energy consumption reduction is a key to upgrade their return of investment (ROI). Energy efficient scheduling on the servers is one of the challenging problems in a dynamic environment such as cloud domain for reducing the energy consumption.

6. the maximization of the reliability of the system: scheduling algorithms considering reliability as part of their objectives must put mechanisms in place to ensure that the workflow execution is completed within the users' QoS constraints even if resource or task failures occur. Or simply minimize possible failure during workflow execution.

Optimisation strategy: the optimisation strategies considered here are heuristic and metaheuristic. Heuristic algorithms deal with particular problems and are designed to find an approximate solution in an acceptable time frame. For instance, a heuristic approach uses

the knowledge about the characteristics of the cloud as well as the workflow application in order to find a schedule that meets the user's QoS requirements. The main advantage of heuristic based scheduling algorithms is their efficiency in terms of performance. They are also easier to implement and more predictable than meta-heuristic based methods. Whereas, meta-heuristic algorithms are high-level and problem-independent algorithms which have a set of rules or strategies to find a globally optimal solution to a given problem. Unlike heuristic-based algorithms, meta-heuristic approaches are generally more computationally intensive and take longer to run. However, they also tend to find more desirable schedules as they explore different solutions using a guided search. Using meta-heuristics to solve the workflow scheduling problem in cloud environments involves challenges such as modelling a theoretically unbound number of resources due to the uncertainty nature of inherent to the cloud. It is also important when using meta-heuristics to define operations to avoid exploring invalid solutions (e.g. data dependency violations) to facilitate convergence, and pruning the search space by using heuristics based on the cloud resource model [3].

1.3.2 Overview of workflow scheduling algorithms

Given that utility computing describes a business model for on-demand delivery of whatsoever consumers need in return for the required remuneration from the supplier, and agreement must be done between the customer and the supplier. That agreement is called Service Level Agreement (SLA). Once the SLA contract is concluded, both parties must respect their commitments, in order to avoid penalties. The requirements of both parties are described in the section 1.3.1. And the requirements considered in this thesis are minimization of execution cost and time as far as user concern, and minimization of energy consumption far as supplier concern.

Therefore, a scheduling algorithm which the goal is to assign the tasks of the workflow to cloud resources (here VMs), must strive to do it in the respect of both users' and providers' requirements.

Many researchers have proposed different approaches based on heuristics or meta-heuristics to solve the problem of single workflow scheduling in distributed systems. We are not doing an exhaustive state of the art here, but we are mainly interested in the algorithms recently proposed and which are currently widespread in cloud computing.

Several approaches exist in the literature to address multi-objective workflow scheduling using meta-heuristic techniques like bio-inspired and nature-inspired algorithms. One popular way of solution for NP-complete problems is the usage of meta-heuristic techniques like bio-inspired and nature-inspired algorithms. Meta-heuristic algorithms are high-level and problem-independent algorithms. For instance Ndam Njoya et al. [73] used an evolutionary-based approach to address the problem of target coverage in wireless sensor networks.

To solve the workflow scheduling problem, several bio-inspired and nature-inspired algorithms were proposed:

Chen and Zhang [74] proposed a meta-scheduler that uses Ant Colony Optimization (ACO) to satisfy user QoS constraints. Hu et *al.* [75] proposed the Immune Particle Swarm Optimization (IPSO) algorithm, an extension of PSO which can diminish the local extreme and avoid the prematurity convergence present in the PSO algorithm. IPSO addresses the problem of multi-objective optimization in dynamic and heterogeneous grid environments by the usage of an objective function based on the satisfaction rate. Chuang et *al.* [76] proposed C-PSO, a variant of PSO that is inspired by the catfish effect observed by Norwegian fishermen when catfishes were introduced into a holding tank of sardines. Pandey et *al.* [77] for the first time used Particle Swarm Optimization (PSO) for workflow scheduling on cloud resources, with the objective to minimize makespan and cost (composed of computation cost and the data transmission cost). Rodriguez and Buyya [78] developed a meta-heuristic optimization technique based on the Particle Swarm Optimization (PSO) to minimize the overall workflow execution cost while meeting the deadline constraint in clouds. Their approach considers the fundamental features of IaaS providers such as the dynamic provisioning and heterogeneity of unlimited computing resources as well as VM performance variation. Elsherbiny et *al.* [79] proposed IWDC, an algorithm extending the natural-based Intelligent Water Drops (IWD) algorithm that optimizes the scheduling of workflow on the cloud. Through simulations, the authors showed that IWDC has noticeable enhancements in the performance and cost in most situations over well-known scheduling algorithms, namely MIN-MIN, MAX-MIN, round-robin, FCFS, and MCT, and over PSO and C-PSO. Verma and Kaushal [80] proposed a non-dominance sort based Hybrid Particle Swarm Optimization (HPSO) algorithm to handle the workflow scheduling problem with multiple conflicting objective functions on IaaS clouds. Ismayilov et *al.* [30] proposed a novel prediction-based dynamic multi-objective evolutionary algorithm, called NN-DNSGA-II algorithm, that incorporates an artificial neural network with the non-dominated sorting genetic algorithm (NSGA-II algorithm). Their model considers the resource failures and changes in the number of objectives as main sources of dynamism.

Even though meta-heuristic strategies produce acceptable results, they are usually time-consuming algorithms due to their relatively long iterative process [81]. Furthermore, the dynamicity of cloud services is a real issue because of the uncertainty during the processing, which may incur more time consumption if meta-heuristic strategies are considered.

Makespan/Deadline aware heuristic algorithms

For the heuristics, one of the most widespread techniques is the list based scheduling, and the most famous and widely used list-based scheduling algorithm is the Heterogeneous Earliest

Finish Time (HEFT) proposed by Topcuoglu et *al.* [82]. HEFT aims to minimize the makespan of workflow execution in heterogeneous environments. It firstly sorts the tasks of the workflow into a scheduling list and then assigns each task to the resource which can finish it earliest in the order of the list. The determination of the finish time takes into account the already mapped tasks to each VM and find (with insertion base) the first time slot that can accommodate the task according to the precedence constraints in the workflow.

Makespan/Deadline and Cost/Budget aware heuristic algorithms

Poola et *al.* [83] present a robust and fault-tolerant scheduling algorithm with resource allocation policies, namely Robustness-Cost-Time (RCT) and Robustness-Time-Cost (RTC). Their algorithm tries to maximize robustness and minimize costs and the total elapsed time (makespan). Their policies maximize the robustness proportionality to the user-defined budget with a reasonable increase in the processing cost. Zheng and Sakellariou [84] proposed a Budget and Deadline Constrained (BDC) scheduling algorithm named BHEFT, which is an improvement of HEFT. BHEFT aims to find a feasible plan for the execution of the workflow allowing providers to decide whether they can agree with the user under his defined deadline and budget, in order to avoid SLA violation. BHEFT is only suitable for Grid since it works under reservation and billing of a fixed number of resources. Verma and Kaushal [85] proposed a cloud-oriented algorithm, namely, Budget and Deadline constrained Heterogeneous Earliest Finish Time (BDHEFT), which is also an extension of HEFT. For each task, BDHEFT generates a BDC schedule plan using the six variables: Spare Workflow Budget (SWB), Spare Workflow Deadline (SWD), Current Task Budget (CTB), Current Task Deadline (CTD), Budget Adjustment Factor (BAF) and Deadline Adjustment Factor (DAF). Three of those variables, namely SWB, CTB, and BAF been inspired by BHEFT [84]. The BDHEFT algorithm proved to be more effective than BHEFT under the same deadline and budget constraints. Abazari et *al.* [86] proposed the multi-objective workflow scheduling (MOWS) algorithm aiming at increasing the security and minimizing the makespan. They considered task interaction issues as a security threat and designed a new systematic method to reduce completion time while ensuring task security demands by introducing the task security sensitivity measurement to quantify tasks security requirements.

Vahid Arabnejad et *al.* [81] proposed Budget Deadline Aware Scheduling (BDAS). BDAS partitions workflow tasks over the different levels, such that each level is seen a bag of tasks (BoT) containing a set of independent tasks, and distributes the user-defined deadline and budget among the levels. While the deadline of each level is calculated before the task selection phase, they used the "All in" Budget Distribution for the budget in which the sub-budget spend by one layer is first resolved before the determination of the budget to use in the next level.

For task selection, the Earliest Start Time (EST) is used as a priority among tasks in each level, such that all the tasks of a level are scheduled before those of the next level. For each task, BDAS performs instance selection using a cost-time trade-off function (CTTF) to select the instance with the highest CTTF.

Unlike the just mentioned heuristics [81, 83, 84, 85], their are other recent workflow scheduling algorithms with dynamic provisioning:

Vahid Arabnejad et *al.* proposed two scheduling algorithms [87] with dynamic provisioning aiming at reducing the cost of computation with a deadline constraint. The two algorithms, namely Proportional Deadline Constrained (PDC) and Deadline Constrained Critical Path (DCCP), are the extended version of two of their previous work. Like BDAS, PDC and DCCP use a preprocessing step for tasks partitioning in which the tasks are partitioned into different levels based on their respective dependencies. Afterwards, the user-defined deadline δ is shared among the different levels, and each level gets its own level-deadline which is also assigned to all the tasks at the level. Then, the PDC algorithm uses a task selection based on a prioritization with eight possible policies (Upward Rank, Downward Rank, Sum Rank, Minimum Execution Time, Maximum Execution Time, Random, Earliest Completion Time and Earliest Deadline First). It finally uses a trade-off function for the selection of the VM of each task. The DCCP algorithm meanwhile determines the constrained critical path (CCP) (presented in [88]) in the workflow based on HEFT upward rank and downward rank. DCCP ends with the Instance Selection phase in which it identifies the most appropriate instance to execute each CCP, all the tasks in a CCP been executed in the same VM. Both PDC and DCCP take into account the dynamic provisioning of VM according to specific situations.

Singh et *al.* [45] proposed the Partition Problem based Dynamic Provisioning and Scheduling (PPDPS) algorithm to minimize the execution cost in the respect of the deadline. PPDPS consists of two main phases, firstly the determination of the VM speed, and secondly the dynamic provisioning and Task-to-VM mapping. In the first phase, they divide the tasks into Bag of Tasks (bots) and assign sub-deadline to each bot, and with the k-means clustering technique applied over the set of lengths of all the paths in the workflow they determine a set of instances type (Rd) to use for the provisioning. In the second phase, each time none of the already leased VMs is able to match to mapping condition they launch the VM provisioning. The VM provisioning begins with the fastest instance, and alternatively, an instance of VMs having higher speed (from the middle of set Rd) and slower speed (from the beginning of set Rd) is chosen. The Task-to-VM mapping is done by BoT.

Faragardi et *al.* [44] proposed a resource provisioning mechanism and a workflow scheduling algorithm based on HEFT, named Greedy Resource Provisioning and modified HEFT (GRP-HEFT). The greedy resource provisioning mechanism lists the instance types according to their

efficiency ratio (its capacity divided by its cost) and takes the most efficient instances constrained by the user defined-budget. The best configuration of instances type with the number of VMs of each type to create is employed on the modified HEFT to obtain the scheduling plan.

Zhu et *al.* [89] proposed a deadline constrained workflow scheduling algorithm (called DyDL) aiming at optimizing the cost of workflow execution with deadline constraint. They introduced for the first time the multi-resource packing to the workflow scheduling in clouds, which has the ability to reduce both the monetary cost and the running time by eliminating idles time slots of VM. Pan et *al.* [90] proposed the Critical-Path-Duration-Estimation based (CPDE) VM Selection strategy aiming at minimizing the execution Cost in respect of the user deadline. They considered the IaaS cloud platform with fluctuating VM performance and used an ARIMA time series model as the underlying prediction method for processing time-fluctuating performance.

Energy aware heuristic algorithms

Among the algorithms proposing energy-efficient techniques we have:

Kimura et *al.* [42] that proposed a slacking algorithm that uses the non-critical path to extends the task execution time by reclaiming slack time to save energy. Huang et *al.* [91] present an enhanced Energy-Efficient Scheduling (EES) algorithm which reduces energy consumption while meeting the performance-based requirements.

Then, Durillo et al. [92] proposed an extended version of the HEFT algorithm denoted multi-objective heterogeneous earliest finish time (MOHEFT) aiming at providing suitable trade-offs between makespan and energy consumption.

Furthermore, Huang et *al.* [91] also proposed two algorithms extending the HEFT algorithm by introducing the energy awareness, called the Enhancing Heterogeneous Earliest Finish Time (EHEFT) and the Enhancing Critical Path on a Processor (ECPOP), and addressed the time and energy-efficient workflow scheduling. Tang et *al.* [23] introduce the DVFS enabled Efficient energy Workflow Task Scheduling (DEWTS) algorithm to obtain more energy reduction. However, the last two algorithms reserve a set of VM instances for the whole makespan.

Then, Li et al. [22] proposed cost and energy-aware scheduling (CEAS) algorithm to minimize the execution cost of workflow and reduce the energy consumption while meeting the deadline constraint in the cloud environment. CEAS first uses a VM selection algorithm that applies the concept of cost-utility to map tasks to their optimal virtual machine (VM) types by the sub-makespan constraint. Afterwards, it employs two tasks merging methods to reduce execution cost and energy consumption. In order to reuse the idle VM instances which have been leased, it further proposed a VM reuse policy. And finally, it utilized a scheme of slack time reclamation to save energy on leased VMs.

More recently, Ritu Garg et al. [21] proposed the Reliability and Energy Efficient Workflow scheduling (REEWS) algorithm. The aim of their proposal is to minimize the energy consumption and maximize the reliability of the workflow execution in the respect of the user-specified QoS/deadline constraint. The REEWS algorithm consists of four main steps: the prioritization of the tasks; the tasks clustering; (user-defined) deadline distribution among the workflow tasks and the mapping of cluster tasks to processors at suitable voltage/frequency levels in order to maximize the overall reliability of the system and minimize of energy consumption.

Singh et al. [93] proposed a meta-heuristic called energy efficient workflow scheduling (EEWS) algorithm, aiming at minimize makespan and energy consumption. EEWS is inspired from hybrid chemical reaction optimization (HCRO) algorithm, and adds a new operator called on-wall pseudo-effective collision to exploit the benefits of swap mutation, and consider dynamic voltage scaling (DVS) along with a novel proposed measure to calculate the amount of energy that can be conserved.

Neha Garg et al. [18] proposed the energy and resource efficient workflow scheduling (ERES) algorithm, which aims at minimizing energy consumption, maximizing resource utilization, and minimizing workflow makespan. The ERES algorithm uses VM migration to deploy/un-deploy the VMs based on the workflow task's requirements and a double threshold policy to perceive the server' status (overloaded/underloaded or normal). ERES also makes use of the DVFS technique.

Conclusion

Even though meta-heuristic strategies produce acceptable results, they are usually time-consuming algorithms due to their relatively long iterative process [3, 81]. Furthermore, the dynamicity of cloud services is a real issue because of the uncertainty during the processing, which may incur more time consumption if meta-heuristic strategies are considered.

It was recently found that few heuristic-based workflow scheduling algorithms consider both deadline and budget constraints at the same time in cloud [81], how much more with energy consumption minimization.

The efforts for improving energy efficiency in IT infrastructures has been a major concern these last years. Even though there are multiples causes of energy consumption, like for instance the usage of cooling equipment and power delivery infrastructure, half of the data centers energy is wasted mostly due to the inefficient management of servers resources [2, 17]. It has been proved that the intelligent management of computing resources can significantly boost the reduction of energy consumption of a system without performance requirements degradation.

Some energy consumption reduction techniques have been presented. Namely, the switching off of idle servers, the VMs/workload consolidation, and the DVFS.

The Dynamic Voltage and Frequency Scaling (DVFS) is the technique employed in this thesis. The DVFS has been made possible in news computer architectures by enabling the software level to control CPU power consumption through the management of the CPU "gearbox", and has as effect the dynamic variation of the power, and therefore of the energy consumption. It is therefore important to know when and how to take advantage of these technique when scheduling workflow tasks because of their great complexity.

In regard to the above-mentioned complexity of both cloud environment and workflows structures, it is essential to design scheduling algorithms tailored for scientific workflows in order to take more advantage of clouds assets [4, 41]. Although the cloud has several advantages, like for instance its flexibility and elasticity, inefficient usage of resources and high computing costs may result if inadequate scheduling and provisioning decisions are made [94].

In this chapter, we have laid the foundation for the general understanding of this report, by presenting the necessary concepts of cloud computing, workflows and workflow scheduling.

The aim of this thesis is the proposition of heuristics for an energy-efficient workflow scheduling in IaaS Clouds, with the minimization of execution cost and time, constrained by the user-defined budget and deadline.

Due to our incremental methodology, the contributions of this thesis are presented in the following three chapters: a cost-time efficient strategy with static provisioning, then a cost-time efficient strategy with dynamic provisioning, and finally three energy and cost-time efficient strategies with dynamic provisioning.

CHAPTER 2

Cost-time trade-off efficient workflow scheduling (CTTWS)

Several workflow scheduling works aim at optimizing the makespan and the budget. However, more investigations are needed for appropriate resources choosing in the large set of instance types offered in cloud environments. This chapter presents a new scheduling algorithm called Cost-Time Trade-off efficient Workflow Scheduling (CTTWS), which consists of four main steps: task selection, Implicit Requested Instance Types Range (IRITR) evaluation, spare budget evaluation, and VM selection. The IRITR evaluation is a novel scheduling concept, which aims at determining a range of VMs instance types that best suits the workflow execution, in order to avoid overbidding and underbidding that may lead to budget and deadline violation respectively. Comparative simulations results against a state-of-the-art algorithm, supported by a Student's T-test, proved that CTTWS can produce better success rates to meet users' deadlines and budgets up-to 38.4% according to the variety of available instance types. This confirms that paying attention to the type of resources is vital.

Introduction

Although the cloud has several advantages, like for instance its flexibility and elasticity, inefficient usage of resources and high computing costs may result if inadequate scheduling and provisioning decisions are made. For example, it is very important to determine the types and the number of appropriate resources and ensure good workload management in order to avoid energy wastage and Service Level Agreement (SLA) violation when running workflow tasks.

In an unknown and diversified market, the assistance of a good sales consultant is important to tailor the quality of purchases to the customer needs and budget. In a cloud, this problem of optimizing the budget-quality ratio becomes the problem of optimizing the execution time and the computing cost according to the user-defined budget and deadline, and the role of a good sales consultant is played by a good scheduling algorithm.

In this chapter, we propose a new workflow scheduling algorithm that aims at optimizing execution time and processing cost, the Cost-Time Trade-off efficient Workflow Scheduling (CTTWS). The CTTWS scheduling algorithm uses a novel concept, the Implicit Requested

This chapter is derived from: **J. E. Ndamlabin Mboula**, V. C. Kamla, and C. Tayou Djamegni: *Cost-time trade-off efficient workflow scheduling in cloud.* Simulation Modelling Practice and Theory 103 (2020): 102107. Elsevier.

Instance Types Range (IRITR) evaluation, to determine a range of VMs instance types that best suits the workflow execution, in order to avoid overbidding as well as underbidding that may lead to budget and deadline violation respectively. Thereby, root tasks are executed on relatively fast instances that speed up execution, with the ability to be reused, and no task uses a slower instance than those in the IRITR. Our algorithm also uses new trade-off factors between time and cost to determine the most viable schedule, and uses this to get the most appropriate type of VM instance to provision. In this work, our trade-off function and its related issues, namely task selection and sparse budget evaluation, are based on a fine granularity approach, compared to their counterparts in the Budget Deadline Aware Scheduling(BDAS) algorithm [81], one of the most recent published work related to our goal and conditions, that rely on a big granularity approach.

2.1 Modelling of the workflow scheduling problem

In this section we present the cloud resource model, the workflow model, and the problem formulation.

2.1.1 Cloud computing model

Our study is limited to a single data center of a public cloud provider. The cloud data center model is similar to the one offered by Amazon EC2 [95]. We assume that the data center is equipped with a set of K types of heterogeneous VM instances, denoted by VMIT= $\{vmit_1, vmit_2, ..., vmit_k, ..., vmit_K\}$, having various processing costs, performances and configurations. Each instance type $vmit_k$ is defined by its computing performance p_k in millions instructions per second (MIPS), its processing cost per billing period c_k and communication bandwidth b_k. An instance with a higher computing performance is . For sake of simplicity, we assume that the communication bandwidth between the instances is uniformly distributed, and denoted by β.

At any moment we consider that P VMs ($VMS = \{vm_1, vm_2, ..., , vm_p, ..., vm_P\}$), each been of a single instance type listed in VMIT, are leased as subscription-based services in a pay-per-use model and are charged per billing period of length τ. A billing period is one hour per VM usage for most IaaS providers; each partial hour consumed being rounded up to a full hour, such that 1 hour and 1 minute (61 min.) will be considered as 2 hours (120 min.) of utilization. As it is often the case [96, 97], we assume that all the $vmit_k$ ∈VMIT are ordered according to their characteristics, such that the order of magnitude of the prices varies accord-

ing to the order of the computing performances as stated in equations (2.1) and (2.2).

$$p_1 < p_2 < ... < p_k < ... < p_K, \tag{2.1}$$

$$c_1 < c_2 < ... < c_k < ... < c_K, \tag{2.2}$$

2.1.2 Workflow model

Workflow as a Directed Acyclic Graph (DAG)

The most commonly used model for scientific application is workflow represented as a Directed Acyclic Graph (DAG). A DAG is a graph $G(WT, E)$ where $WT = \{t_1, t_2, ..., t_n\}$ is the set of the tasks of the workflow (the weight of task t_i, in terms of millions of instructions, is denoted by Z_i), and $E = \{e_{i,j} = (t_i, t_j) | 1 \leq i, j \leq n, i \neq j\}$ a the set of edges representing the existing data and control dependencies between tasks. Thus $e_{i,j} \in E$ if there is a precedence constraint between t_i and $t_j \in WT$, such that the execution of t_j can start only after t_i finishes its execution and sends data (of size s_{ij} in Megabytes (MB)) to t_j. Task t_i is a parent of t_j and t_j a child of t_i. A task is ready to start its execution when all of its parents have been executed and all its required data have been provided. Any task with no parent is an entry task and any task with no child is an exit task. The set of the parents (resp. children) of the task t_j is denoted $pred(t_i)$ (resp. $succ(t_i)$). The Critical Path (**CP**) of a DAG is defined as its longest path.

A workflow can have one or more entry tasks (tasks without parent), and one or more exit tasks (tasks without child). Entry tasks and exit tasks are denoted as t_{entry} and t_{exit} respectively.

Definitions

In this section, we define the execution time and the processing cost of a submitted workflow, consisting of a total of n tasks, stemming from the workflow model and cloud computing model.

Given a task t_i, its execution time on a resource of instance type $vmit_k$ is denoted by $ET(i, k)$ and defined as

$$ET(i, k) = \frac{Z_i}{p_k}, \tag{2.3}$$

and the data transfer time from t_i to $t_j \in succ(t_i)$ denoted by $TT(i, j)$ is defined as

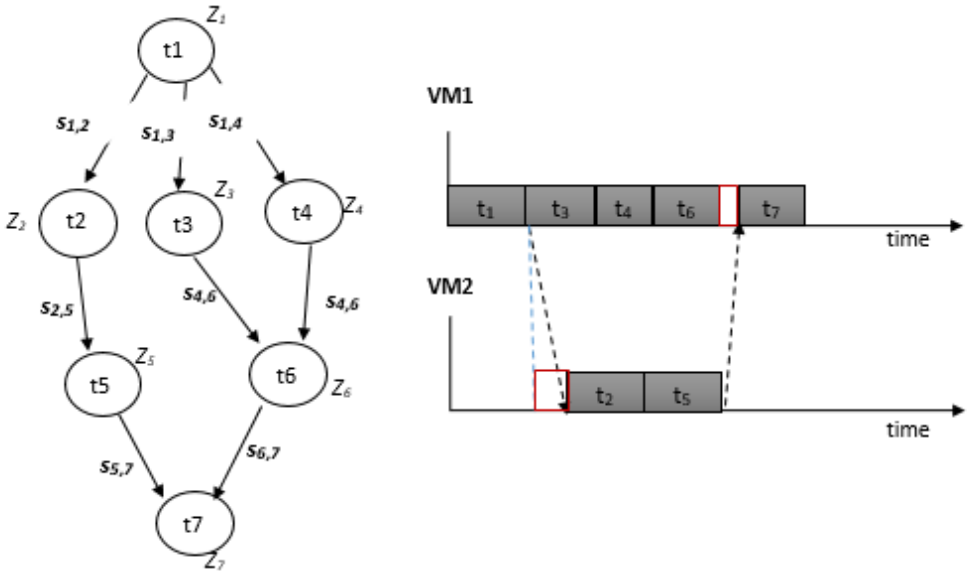


Figure 2.1: Example of mapping of DAG tasks onto VMs, with idle time slots due to data transfer

$$TT(i,j) = \begin{cases} 0 & , \ if \ t_i \ and \ t_j \ are \ mapped \ of \ the \ same \ VM \\ \frac{s_{ij}}{\beta} & , \ otherwise \end{cases} \tag{2.4}$$

The execution cost of task t_i on a resource of instance type $vmit_k$ is denoted by $EC(i,k)$ and defined as

$$EC(i,k) = \left\lceil \frac{ET(i,k)}{\tau} \right\rceil \times c_k, \tag{2.5}$$

where τ is the length of a billing period.

Assuming that a number of tasks (t_i, $i \in I$) are mapped onto the same VM instance vm_p, supposed to be of instance type $vmit_k$, the cumulative execution cost will be evaluated as follow:

$$EC(I,k) = \left\lceil \frac{\sum_{i \in I}\{ET(i,k)\} + TITS_p}{\tau} \right\rceil \times c_k, \tag{2.6}$$

where $TITS_p$ (Transfer Idle Time Slots) is the sum of the idle time slots due to data transfer awaited by the VM vm_p. The example presented in Figure 2.1 gives an illustration of TITS for the mapping of a DAG of seven tasks onto two VMs. The red rectangles are TITS.

We denote the Earliest Start Time and Earliest Finish Time of task t_i as $EST(t_i)$ and $EFT(t_i)$ respectively. They are defined as

$$EST(t_i) = \begin{cases} 0 & , \ if \ t_i = t_{entry}, \\ \max\limits_{t_j \in pred(t_i)} \{EFT(t_j) + TT(j,i)\} & , \ otherwise \end{cases} \tag{2.7}$$

and

$$EFT(t_i) = EST(t_i) + ET(i,K), \tag{2.8}$$

where $ET(i,K)$ is the minimum possible ET of task t_i over all possible VM instance types, according to the relation (2.1).

In like manner, we also consider the Actual Start Time and the Actual Finish Time of a task t_i mapped onto a VM vm_p^k of instance type $vmit_k$. They are denoted by $AST(t_i, vm_p^k)$ and $AFT(t_i, vm_p^k)$ respectively. Their values may be different from those of $EST(t_i)$ and $EFT(t_i)$, due to the heterogeneity of VMs and the fact that functions EST and EFT provide the minimum possible value over all possible mapping of tasks onto VMs. The workflow completion time (also called makespan), denoted by M_G, is defined as the Actual Finish Time of the exit task t_{exit}. The minimum makespan, denoted by $minM_G$, is defined as the Earliest Finish Time of t_{exit}. Thus, $minM_G = EFT(t_{exit})$.

Finally, the total cost of executing a workflow is the sum of the execution cost of all the tasks when effectively mapped onto VMs, and defined as:

$$Cost_G = \sum_{t_i \in G} EC(i, map(i)), \tag{2.9}$$

where $map(i)$ denotes the VM on which task t_i is mapped, $1 \leq p \leq P, \ 1 \leq k \leq K$.

2.1.3 Problem formulation

In this chapter, the VMs provisioning strategy is the static one. It is considered that P resources of different instance types are available. Given these P provisioned resources, the workflow scheduling aims at determining the optimal execution order of tasks and task mapping onto VMs with respect to user and workflow constraints. Here, user constraints are the user-defined deadline δ, assumed to be great or equal to the minimum makespan ($\delta \geq minM_G$), and the user-defined budget B, assumed to be great or equal to the total cost provided by equation (2.9). The workflow constraints are made of task weights and data dependencies between them. We assume that workflows are submitted to the cloud by users.

The question to deal with is: *how to build a workflow scheduling algorithm able to reduce*

execution cost and execution time in the respect of the user-defined budget and deadline?

The problem can be formulated as a mathematical optimization problem:

$$\begin{cases} Reduce(M_G) \\ Reduce(Cost_G) \\ Subject\ to\ M_G \leq \delta\ and\ Cost_G \leq B \end{cases} \tag{2.10}$$

2.2 The Proposed Scheduling Algorithm: CTTWS

In this section, we present our proposed solution for the workflow scheduling problem, the Cost-Time Trade-off effective Workflow Scheduling (CTTWS), which aims at optimizing both processing costs and times. Our scheduling algorithm has four main steps summarized in Table 2.1. The steps are not necessarily logically ordered in the table.

Table 2.1: The four main steps of CTTWS algorithm

	Step	**Description**
2.2.1	Spare Budget Evaluation	Due to the cloud billing period model, mapping a new task can leave the current overall cost unchanged. Thus, only the additional cost induced by the mapping of a new task is reduced from the current spare budget.
2.2.2	Implicit Requested Instance Types Range (IRITR) evaluation	By the usage of the minimum makespan $minM_G$ and the user-defined budget (B) and deadline (δ), we determine the VMs instance types range that best suits the workflow execution throughout.
2.2.3	Task Selection	Each task is selected in the ready list based on its priority (EST Asc) for execution
2.2.4	VM Selection	In this phase, we introduce a new Cost-Time trade-off policy. We find the best VM in regard of the IRITR, the combination of the cost and time, and the structure of the workflow.

2.2.1 Spare Budget Evaluation

Due to the cloud billing period model, mapping a new task can leave the current overall cost unchanged. Thus, only the additional cost (also called extra cost) induced by the mapping of a new task onto a VM is reduced from the current spare budget. The spare budget is defined as the amount of money remains after mapping a task onto a VM. The starting spare budget is equal to the user-defined budget. When a new task is mapped onto a VM, the extra cost is subtracted from the current spare budget. Note that, the extra cost is zero if the new task

is scheduled in the current period. A new billing period is started if the execution of the new task can not take place in the current period.

Our spare budget management strategy is based on fine-grained updates, as an update is done when mapping a single task onto a VM, unlike the one introduced in BDAS [81] which is based on big-grained updates. BDAS updates the spare budget after mapping all the tasks of the same level onto VMs, assuming that the workflow is partitioned into levels composed of non-dependent tasks.

2.2.2 Implicit Requested Instance Types Range (IRITR) evaluation

The computing power of different types of resources is generally proportional to their price, as in Amazon EC2 [96]. As users and cloud providers know, the cost of processing plays a very important role. It is based on the billing model, the planning algorithm, and the workflow structure. In an unknown and diverse market, if we do not have a good sales consultant, it will be difficult to optimize the quality of purchases against the budget. In a cloud, this problem of optimizing the budget-quality ratio becomes the problem of optimizing both cost and makespan with respect to both budget and deadline. This last problem is solved here by avoiding both overbidding and underbidding of resources during the execution of the workflow. To solve this avoidance problem, we introduce a new metric, the Implicit Requested Instance Types Range (IRITR), which represents a range of instance types that the budget can afford based on the makespan, deadline, and workflow structure. The IRITR construction process is described in the following.

- The Minimum and Maximum Remaining Number of Billing Period ($MinRNBP$, $MaxRNBP$) are defined respectively by

$$MinRNBP = \left\lceil \frac{remM_G}{\tau} \right\rceil \times remNbTasks; \tag{2.11}$$

and

$$MaxRNBP = \left\lceil \frac{rem\delta_G}{\tau} \right\rceil \times remNbTasks; \tag{2.12}$$

- The Minimum and Maximum Implicit Requested Instance Type Cost ($MinIRITC$, $MaxIRITC$) are defined respectively by

$$MinIRITC = \frac{remB_G}{MaxRNBP}; \tag{2.13}$$

and

$$MaxIRITC = \frac{remB_G}{MinRNBP} \tag{2.14}$$

where $remM_G$ is the remaining makespan, $rem\delta_G$ the remaining deadline, δ the deadline and $remB_G$ the remaining budget, and $remNbTasks$ the number of tasks not yet mapped onto a VM.

After the computation of equations (2.11), (2.12), (2.13) and (2.14), we determine the slowest ($cheapestIRIT$) and the fastest ($expensiveIRIT$) VMs instance types whose the costs are between $MinIRITC$ and $MaxIRITC$, and we set the range as $IRITR = [cheapestIRIT, expensiveIRIT]$.

During the execution of a workflow, the remaining makespan and deadline at a any time t are respectively $remM_G = M_G - t$ and $rem\delta_G = \delta - t$. Since the makespan is unknown during the execution of the workflow, it is approximated by $minM_G$ and the remaining makespan is approximated as follows: $remMG \simeq minM_G - t$. $IRITR$ could be constructed either at any time a task has to be mapped onto a VM (also called VM selection) or only at the beginning of the mapping process. In order to minimize the load of the mapping process, we choose the second. In this case, $remM_G \simeq minMG - AST(t_{first}) \simeq minMG$ and $rem\delta_G = \delta - AST(t_{first}) = \delta_G$ as $AST(t_{first}) = 0$.

It may happen that the user-defined deadline is smaller than the minimum makespan, i.e. $\delta_G < minM_G$, and this implies that $rem\delta_G < remM_G$, which in turn implies that $MinIRITC > MaxIRITC$. This means that the left bound of the $IRITR$ interval is greater than its right bound. Hence a contradiction. This situation occur mainly for data or I/O intensive workflow, due to the expression of the EFT. To handle this, the values of $rem\delta_G$ and $remM_G$ are reset to non-contradictory values: $remM_G = 0.75 \times rem\delta_G$ and $rem\delta_G$ remains unchanged. According to the limited number of VMs used, there will be VM reuse along the line. Hence, this readjustment may always leads to a success.

Another robustness management must be handle if no instance type belongs to the $IRITR$. In that case, it is only one instance type that is chosen. Either $cheapestIRIT$, if all the existing instances are cheaper, or $expensiveIRIT$, if all the existing instances are more expensive.

2.2.3 Task Selection

Tasks are ordered in a list, called ready list, according to their Earliest Start Time (EST) such that all the root tasks receive the highest priority.

In our task selection strategy, a task is ready for execution once its parents have been executed and its awaited input data received. Whereas, in BDAS [81], tasks are executed level

by level, meaning that a task is ready for execution once all tasks in the prior level have been scheduled. This may slow down the execution of the workflow as a task for which all parents in the previous levels have finished their execution is forced to wait on non-dependencies.

2.2.4 VM Selection

Once the workflow tasks have been ordered according to their $ESTs$, they are ready to be mapped onto the available VMs, one after the other. Generally, faster resources are more expensive compared to slower ones. Therefore, there is often an exploitable trade-off [81] between execution time and the cost of resources. The trade-off between cost and time is handled according to the trade-off function defined as follows. Given a task t_i and a VM vm_p, we have

$$CTTF_i^p = Cost_i^p + Time_i^p; \tag{2.15}$$

where $Cost_i^p$ is the cost part and $Time_i^p$ the time part, and they are evaluated according to equations (2.16) and (2.17) respectively.

Cost

The cost part of our trade-off function is defined as the ratio between the spare budget obtained if the task t_i is mapped onto the VM vm_p and the smallest possible spare budget that one can get at the current time:

$$Cost_i^p = \frac{spareB^t - EC(i,p)_{added}}{spareB^t - EC(i)_{min}}; \tag{2.16}$$

where $spareB^t$ is the spare budget at the current time t as described in the sub-section 2.2.1, $EC(i)_{min}$ the minimum cost of performing task t_i among all VMs with respect to their actual load (as given in equation (2.5)), and $EC(i,p)_{added}$ is the additional execution cost if task t_i is mapped onto VM vm_p (also according to equation (2.5)). The value of $EC(i,p)$ is zero if VM vm_p is already provisioned and task t_i can be executed during the current paid billing interval, since each instance is billed on a time slot until it is complete. The lower the added cost($EC(i,p)_{added}$), the higher the cost part ($Cost_i^p$), and the better the budget savings. Thus, formula (2.5) estimates the degree of budget savings of task t_i on VM vm_p.

Time

The time part of the trade-off function is defined as the inverse of the Actual Finish Time ($AFT(t_i, vm_p)$) of task t_i on VM vm_p. So, the lower the AFT, the higher the $Time_i^p$ is. Thus, formula (2.17) estimates the degree of time savings of task t_i on VM vm_p. This is exploited to locate the VM that can perform task t_i as fast as possible based on the current loads and the performances of the available VMs. The $AFT(t_i, vm_p)$ is is determined by finding the first idle time slot capable of holding the task t_i on VM vm_p like in [82].

$$Time_i^p = \frac{1}{AFT(t_i, vm_p)}; \tag{2.17}$$

Equation (2.17) avoids the early convergence of $Time_i^p$ to zero, which is a cause of erroneous selection of a suitable VM [81]. When the deadline is distributed per task level, the way of constructing the time part proposed in [81] often produces a negative time-part value, which in turn produces a wrong result when the deadline is tight, although it would still have been possible to find-out a good mapping.

VM Selection algorithm

The VM selection algorithm pseudo-code is depicted in Algorithm 1.

Algorithm 1 VM Selection Algorithm

Input: The ordered list of the tasks *readyList*, and the list of VMs VMS
Output: All the tasks are mapped to to their suitable VMs

for $t_i \in readyList$ **do**
  if t_i **has already been mapped to a VM then**
    continue; //continue to the next task since this task has been mapped through one of its parents
  end if
  for $vm_p \in VMS$ **do**
    Calculate the $CTTF_i^p$
    $bestVM \leftarrow vm_p$ **if** vm_p **fulfils the following conditions:**
    (C1) vm_p **have the highest** $CTTF_i^p$
    (C2_1) If t_i **is a root task, the VM** vm_p **must be of instance type** $expensiveIRIT$
    (C2_2) If t_i **is not a root task, the VM** vm_p **must be of an instance type belong to** $IRITR$
  end for
  Map t_i **to** $bestVM$
  if t_i **has exactly one parent and one child then**
    $t_{next} \leftarrow child(t_i)$
    while t_{next} adds zero cost on $bestVM$, and has exactly one parent **do**
      Map t_{next} **to** $bestVM$
      $t_{next} \leftarrow child(t_{next})$
    end while
  end if
  $spareB^t \leftarrow spareB^t - EC(i,p)_{added}$
end for

2.2.5 The CTTWS algorithm

Here, we present the CTTWS algorithm. It consists of four main steps as mentioned above. The CTTWS algorithm strives to enable the cloud scheduler to spend less money to complete the workflow without exceeding the deadline. The CTTWS algorithm pseudo-code is depicted in Algorithm 2.

We have noticed that the evaluation of the Implicit Requested Instance Types Range (IRITR) is a good track for the scheduling plan, while the simple use of EST prioritization among workflow tasks avoids scheduling overhead within the paths of the workflow. Since we didn't adopt a level-based deadline/budget distribution, careful management of costs, and deadlines along the line was required because the workflow structure may contain parallel tasks that may have close deadlines. Therefore, our VM selection algorithm policy and our trade-off function are able to reduce both the monetary cost and the deadline.

Algorithm 2 CTTWS Algorithm

Input: The DAG, and the list of VMs VMS
Output: All the tasks are scheduled to their suitable VMs

1: **Order the tasks** $readyList$ **in Asc** EST
2: $[cheapestIRIT, expensiveIRIT] \leftarrow IRITR$ **Evaluation**
3: **Call the VM Selection Algorithm**
4: **for** $t_i \in readyList$ **do**
5: **Schedule task** t_i **to its mapped** vm_p
6: **end for**

2.2.6 Time Complexity

To determine the time complexity of both BDAS and CTTWS algorithms, scheduling phases that must be considered are task selection and VM selection. In fact, it is the phases which contain the deepest nesting of loops, with the parameters of magnitude relating to the problem (n the number of tasks, and P the number of VMs). For the task selection, given a workflow containing n tasks, we need $O(n^2)$ time for the determination of their EST. Afterward, sorting the tasks takes $O(nlogn)$ time complexity, which gives an overall of $O(n^2)$ for the task selection. For each ready task, to select the suitable VM all the VMs should be examined. Which gives a time complexity of $O(n \times P)$ for VM selection, where P is the number of VMs. However, in the case of CTTWS that happens when workflow tasks are very connected since the tasks in the same pipeline are mapped to the same VM at ones.

Hence both algorithms have a time complexity order of $O(n \times (n + P))$. However, since we have $P < n$, we conclude that BDAS and CTTWS algorithms have a polynomial time complexity of $O(n^2)$.

2.3 Performance evaluation

In this section, we present the experiment setup and analyse the simulation results.

We have used the Pegasus workflow generator [5] during experimentation to create the structure of the five real-world scientific workflows (Montage, CyberShake, Epigenomics, SIPHT, and LIGO), in different workloads (the number of tasks of the workflow): 50, 100, 200, 500 and 1000 tasks.

We implemented CTTWS and BDAS, one of the most recent published algorithms [81] related to our field of study, and compared their performances on WorkflowSim simulator [98], an extension of CloudSim [99] for investigating workflows.

2.3.1 Experiment setup

For the simulations we consider the system as a single data center having ten different instance types that are based on the US-east (Ohio) Amazon region [95], collected in July 2019, and which the characteristics are presented in Table 2.2.

Table 2.2: Instance types based on Amazon EC2

Type	vCPU	ECU	Memory(GB)	Cost($)/Hour
m3.medium	1	3	3.75	0.067
m4.large	2	6.5	8	0.10
m4.xlarge	4	13	16	0.20
m4.2xlarge	8	26	32	0.40
m4.4xlarge	16	53.5	64	0.80
m5.8xlarge	32	131	128	1.536
m4.10xlarge	40	124.5	160	2.00
m5.12xlarge	48	173	192	2.304
m4.16xlarge	64	188	256	3.20
m5.24xlarge	96	345	384	4.608

We have configured the simulation environment as follows. The bandwidth between instances is fixed to 20 MBps, the value of the vCPU of each instance is considered as its processing capacity in Million Instruction Per Second (MIPS) as seen in [41]. The charging model has being configured to reflect the Amazon EC2 instances charge that is an hourly interval from the time of provisioning. The virtualization system used is Xen. The VMs have been created such that the number of VMs per instance type is the same.

2.3.2 Performance metrics

We compare the performances of CTTWS and BDAS [81], one of the most recent and relevant scheduling algorithms in our field study, based on the following well-known performance metrics:

Cost Ratio (CR), Time Ratio (TR) and Success Rate (SR).

- Cost Ratio (CR): The CR metric is used to compare the achieved costs of scheduling algorithms. The CR of a scheduling algorithm is calculated by dividing the overall cost (expressed in equation 2.9) by the user-defined budget (B). A CR value greater than 1 indicates a cost larger than the budget, which counts as a failure to meet the defined budget. CR value of less than 1 indicates that the scheduled workflow meets the budget.

$$CR = \frac{Cost_G}{B}; \tag{2.18}$$

- Time Ratio (TR): In a similar way, the TR metric is used to compare the achieved times of scheduling algorithms. The TR of a scheduling algorithm is defined as the ratio between the overall makespan and the user-defined deadline (δ). A TR value greater than 1 indicates a makespan larger than the deadline, which counts as a failure to meet the defined deadline. TR value of less than 1 indicates that the scheduled workflow meets the deadline.

$$TR = \frac{M_G}{\delta}; \tag{2.19}$$

- Success Rate (SR): The SR metric is used to compare the achieved successes of scheduling algorithms. The SR of a scheduling algorithm is defined as the ratio between the number of ran simulations that successfully met both deadline and budget constraints (denoted by NB success), and the total number of experiments (denoted by NB_{Exp}) :

$$SR = \frac{NB_{success}}{NB_{Exp}}; \tag{2.20}$$

  When both the CR and TR values are less than one, that means both the overall execution time and cost meet respectively the user-defined budget and the user-defined deadline. This situation is considered a success.

2.4 Simulation Results and Analysis with Student's T-Test

We conducted two different types of complementary experiments. The first type is based on a variation of the deadline value, the budget value, and the number of VMs, while the second is based on the variation of the instance types among the ones described in Table 2.2.

2.4.1 Experiment Type 1

In this experiment type we calculated for each workflow type the fastest schedule (FS) as a baseline schedule and the lowest budget (LB) as in [81]:

$$FS = \sum_{t_i \in CP} ET(i, K), \tag{2.21}$$

where $ET(i, K)$ is the execution time of task t_i on the fastest instance according to equation (2.3). FS can be viewed as the sum of the minimum execution times of the tasks belonging to the Critical Path (CP).

$$LB = \sum_{t_i \in G} EC(i, 1), \tag{2.22}$$

where $EC(i, 1)$ the execution cost of task t_i on the cheapest instance according to equation (2.5). LB is the lowest possible cost required for executing a workflow, irrespective of the completion time.

Then by using equations (2.21) and (2.22), we set variation ranges for user-defined budget and deadline from tight to moderate to relaxed as follow:

$$deadline = \alpha * FS, \alpha \in [4, 8, 12, 16], \tag{2.23}$$

$$budget = \beta * LB, \beta \in [4, 8, 12, 16], \tag{2.24}$$

For each of the five workflow structures, we consider five different sizes (50, 100, 200, 500, and 1000 tasks), resulting in a total of $5 \times 5 = 25$ workflows. We performed 20 experiments per workflow with the same deadline, budget, and VMs. This corresponds to $5 \times 20 = 100$ experiments per workflow structure. The variation of deadline and budget factors yields 16 different cases per workflow structure. By considering both deadline and budget variations and three set sizes (1000, 5000, and 10000) of VMs, the number of experiments per workflow structure is $100 \times 16 \times 3 = 4800$. Therefore, the overall number of experiments is $5 \times 4800 = 24000$.

For this experiment type the comparative studies were conducted by analyzing both box plots for cost ratio and time ratio (cost and time efficiency), and bar plots for success rate as respectively defined in equations (2.18), (2.19) and (2.20). Because the results per number of VMs were almost similar we just studied the cost and time efficiency for the experiments with 1000 VMs. Box plots are used to show the degree of dispersion of both the cost and time ratios. Box plots are a standardized way of displaying the distribution of data based on the following: the minimum, the first quartile (also called the 25th percentile), the median (also called the

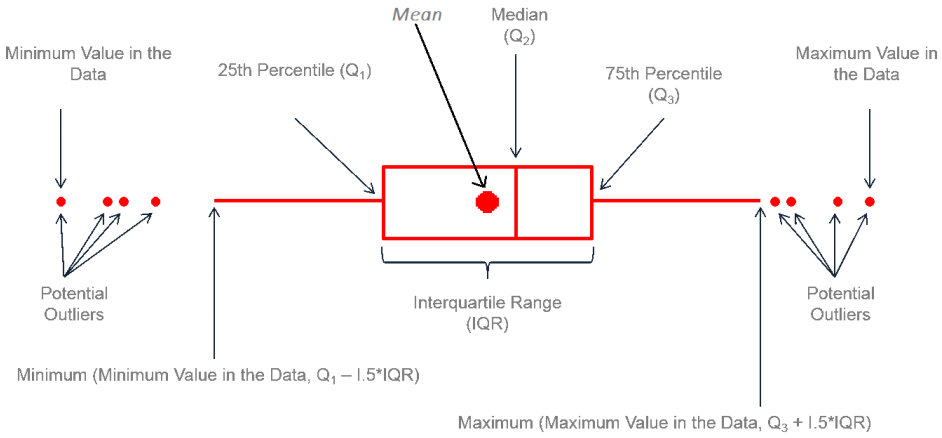


Figure 2.2: Different parts of a box plot.

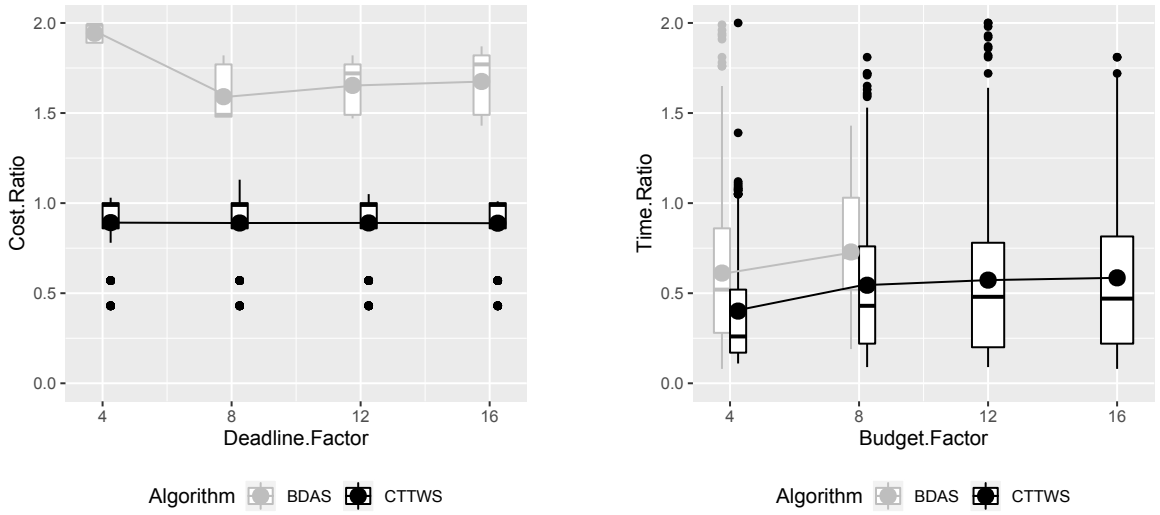
middle value), third quartile (also called the 75th percentile), the interquartile range (which spans from the 25th to the 75th quartile), the maximum and potential outliers. A percentile is a value in a data distribution below which a given percentage of values falls. For example, the 25th percentile (also known as the first quartile) is the value below which 25th of the values fall. If the values of 75% of the data distribution are lower than 1.6, then 1.6 is the 75th percentile. At the 50th percentile or the median, 50% of the values are less than or equal to (resp. are greater than or equal to) that value. Figure 2.2 describes the structure of a box plot[1]. The dot within the interquartile range denotes the mean of the data distribution.

Performance for MONTAGE workflow for Experiment type 1

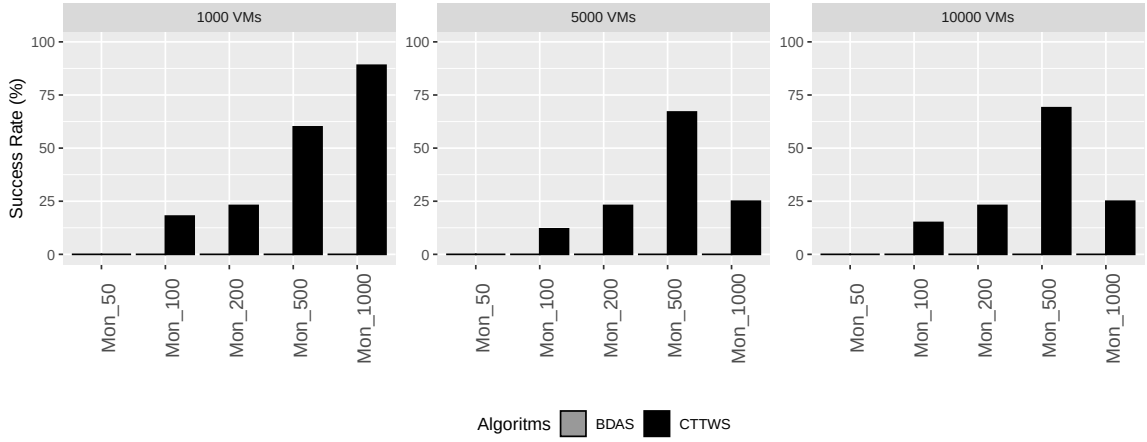
The results obtained for MONTAGE workflow are presented in Figure 2.3, for both cost and time efficiency, and in Figure 2.3c, for success rate. For both cost and time efficiency, CTTWS has better performance than BDAS, by achieving smaller cost and time ratios. Apart when the budget factor, β, is 4 or 8, in which case BDAS has more than 75% of experiments with a good time ratio ($TR \leq 1$) (Figure 2.3b), in all the experiments, BDAS fails in terms of cost efficiency because it does not meet the user-defined budget ($CR > 1$) (Figure 2.3a). Moreover, for CTTWS, more than 75% of the schedules meet the deadline ($TR \leq 1$) for each value of the budget factor (β), and more than 50% of the schedules meet the budget ($CR \leq 1$) (Figure 2.3a) for each value of the deadline factor (α).

In terms of average success rate, BDAS has an average of 0.0% while CTTWS records an average of 29.9% (Figure 2.3c).

[1]https://www.leansigmacorporation.com/box-plot-with-minitab/



(a) Cost Efficiency (Deadline Factor vs. Cost Ratio) (b) Time Efficiency (Budget Factor vs. Time Ratio)



(c) Success rate (%) for MONTAGE workflow with 1000, 5000 and 10000 VMs.

Figure 2.3: Cost efficiency, time efficiency, and success rate (%) of CTTWS vs BDAS for MONTAGE workflow.

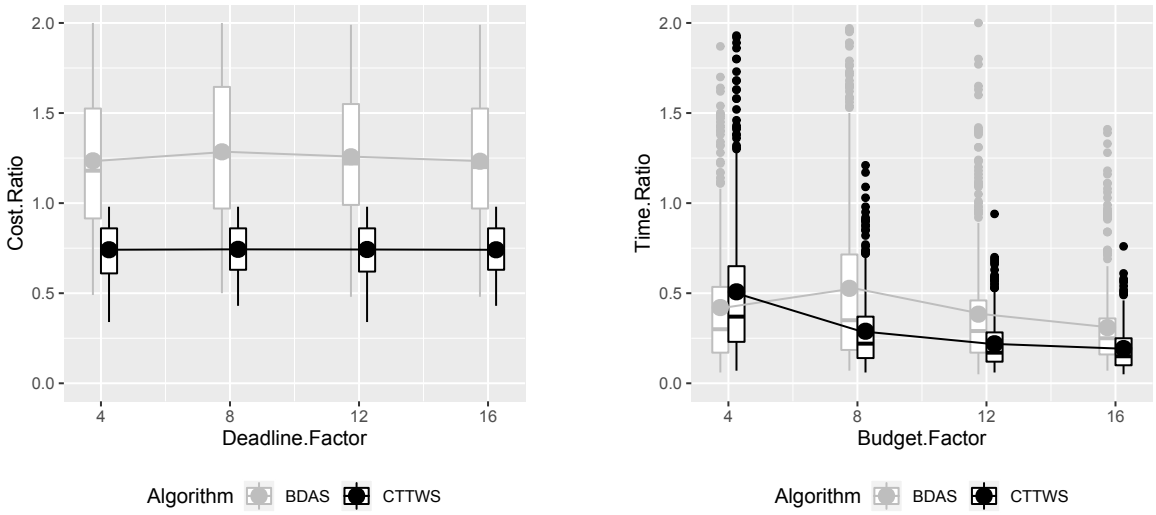
Performance for CYBERSHAKE workflow for Experiment type 1

The results obtained for CYBERSHAKE workflow are presented in Figure 2.4, for both cost and time efficiency, and in Figure 2.4c, for success rate. For CYBERSHAKE workflow, CTTWS has a good Cost Ratio (Figure 2.4a), and fails to achieve the deadline for less than 25% of the cases for which the budget factor β is 4 or 8 (Figure 2.4b). BDAS exceeds the budget in more than 50% of cases (Figure 2.4a).

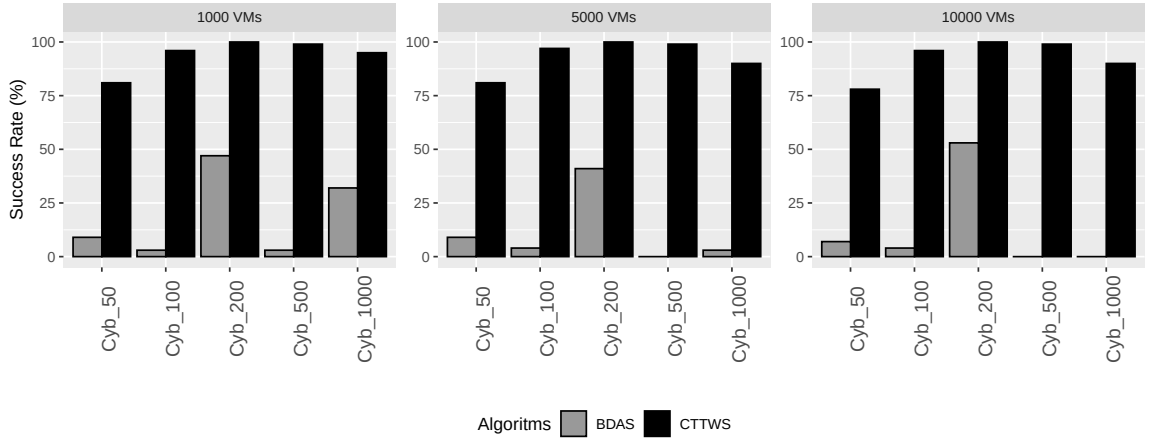
The average success rate is 14.3% for BDAS and 93.4% for CTTWS.

Performance for EPIGENOMICS workflow for Experiment type 1

The results obtained for EPIGENOMICS workflow are presented in Figure 2.5, for both cost and time efficiency, and in Figure 2.5c, for success rate. Even for EPIGENOMICS, CTTWS



(a) Cost Efficiency (Deadline Factor vs. Cost Ratio) (b) Time Efficiency (Budget Factor vs. Time Ratio)



(c) Success rate (%) for CYBERSHAKE workflow with 1000, 5000 and 10000 VMs.

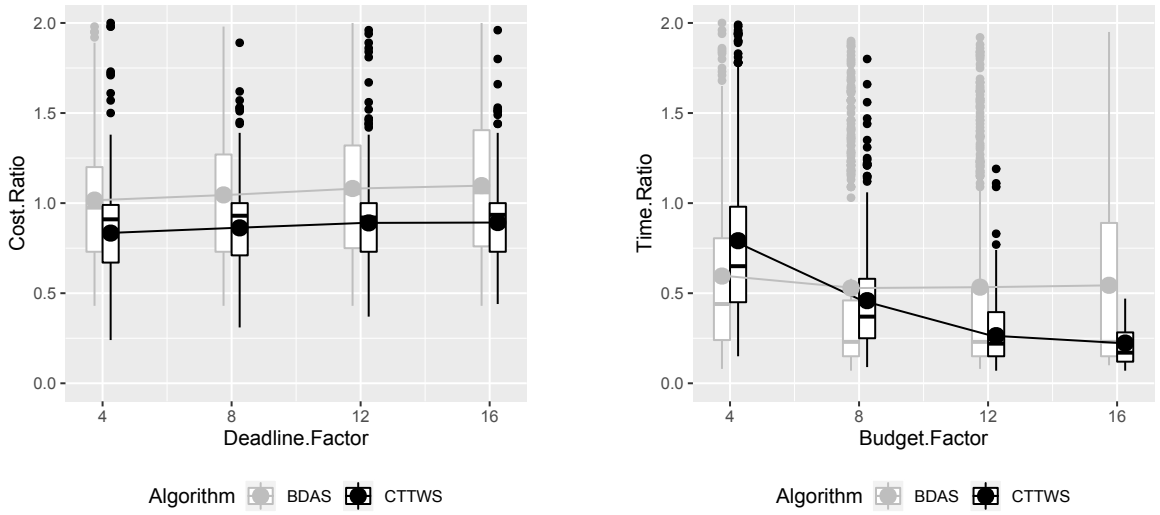
Figure 2.4: Cost efficiency, time efficiency, and success rate (%) of CTTWS vs BDAS for CYBERSHAKE workflow.

outperforms BDAS in terms of cost efficiency (Figure 2.5a) and time efficiency (Figure 2.5b), and therefore in terms of success rate (Figure 2.5c). However, for both algorithms, more than 75% of the schedules meet the deadline ($TR \leq 1$) for each value of the budget factor (β) (Figure 2.5b), and almost 50% of the schedules meet the budget ($CR \leq 1$) (Figure 2.5a) for each value of the deadline factor (α).

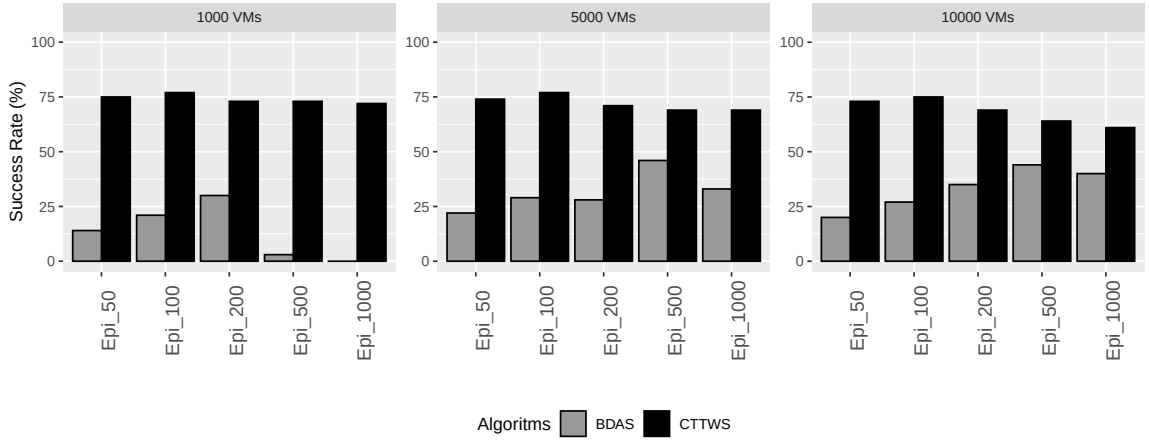
In terms of average success rate, BDAS has an average of 26.1% while CTTWS records an average of 71.5%.

Performance for SIPHT workflow for Experiment type 1

The results obtained for SIPHT workflow are presented in Figure 2.6, for both cost and time efficiency, and in Figure 2.6c, for success rate. For SIPHT workflow, both algorithms achieve



(a) Cost Efficiency (Deadline Factor vs. Cost Ratio) (b) Time Efficiency (Budget Factor vs. Time Ratio)



(c) Success rate (%) for EPIGENOMICS workflow with 1000, 5000 and 10000 VMs.

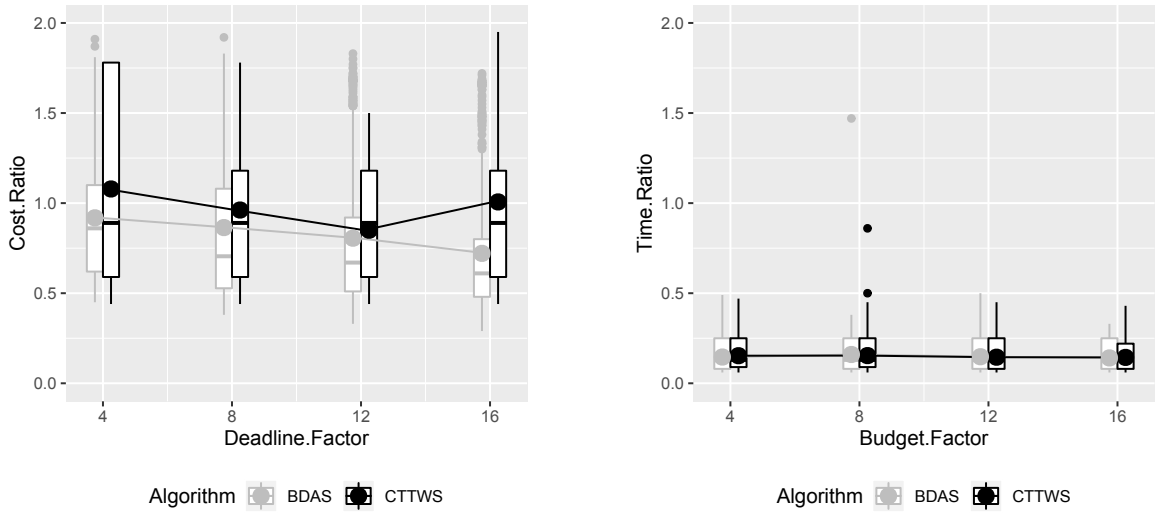
Figure 2.5: Cost efficiency, time efficiency, and success rate (%) of CTTWS vs BDAS for EPIGENOMICS workflow.

the execution in at most half of the deadline (Figure 2.6b) in almost all the cases, whereas BDAS has better budget management than CTTWS. However, as we can see in Figure 2.6a, although CTTWS fails to meet the budget, the values of the ratio of its cost are slightly greater than 1, meaning that the failure is a bit close to success.

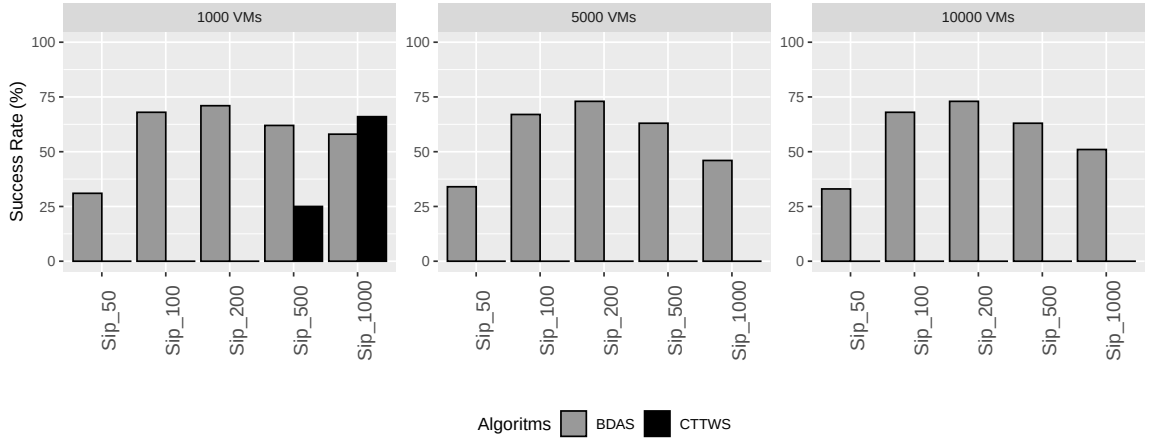
The average success rate is 6.1% for CTTWS and 57.4% for BDAS.

Performance for LIGO workflow for Experiment type 1

In the case of LIGO workflow, except for a tiny percentage when $\alpha = 8$, BDAS always exceeds the budget (Figure 2.7a). However, BDAS have more than 75% of completion time in the deadline (Figure 2.7b); but still leading to almost 100% failure due to budget. Whereas, CTTWS realised more than 50% of planning in the budget (Figure 2.7a), and more than 75%



(a) Cost Efficiency (Deadline Factor vs. Cost Ratio) (b) Time Efficiency (Budget Factor vs. Time Ratio)



(c) Success rate (%) for SIPHT workflow with 1000, 5000 and 10000 VMs.

Figure 2.6: Cost efficiency, time efficiency, and success rate (%) of CTTWS vs BDAS for SIPHT workflow.

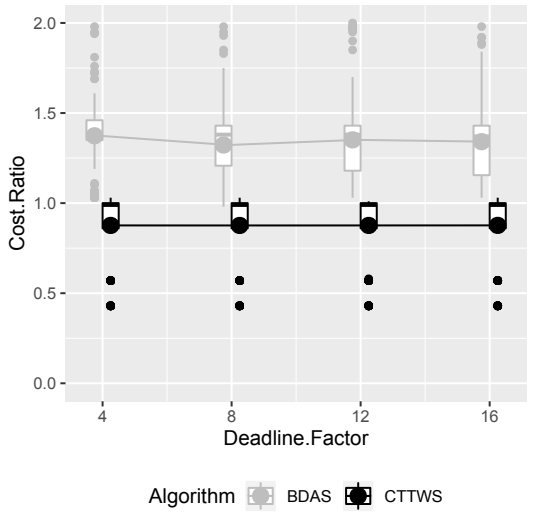
of completion time in the deadline (Figure 2.7b).

The results obtained for LIGO workflow are presented in Figure 2.7 for both cost and time efficiency, and in Figure 2.7c, for success rate.

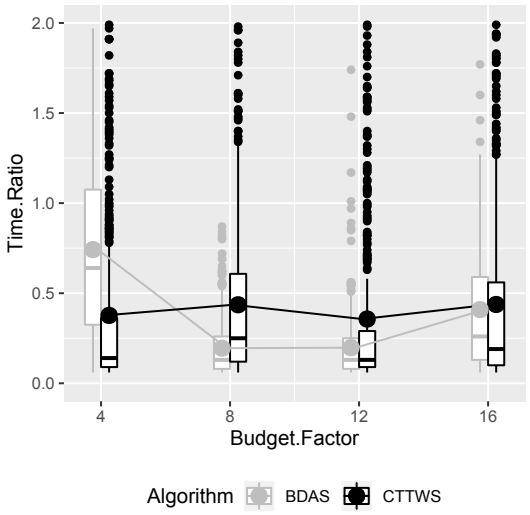
The average success rate is 37.7% for CTTWS and almost 0.2% for BDAS.

Performance summary for Experiment type 1

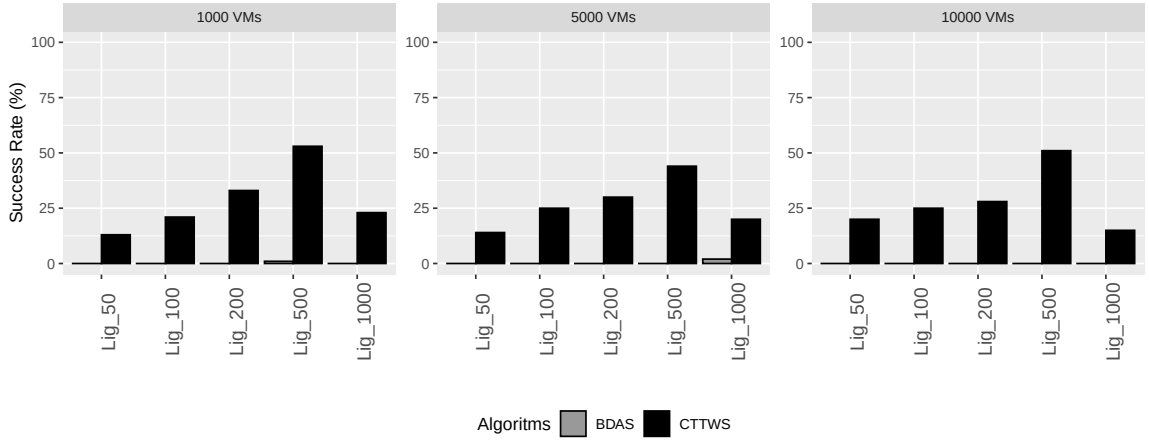
In summary, CTTWS has a global average of SR that is up-to 32.5% higher than the one of BDAS as depicted in Table 2.3; and both algorithms meet deadlines better than budget requirements, which corroborates with what is observed in [81]. Also, it appears that the number of available VMs (at least from a certain threshold) doesn't have a significant impact on the performance of both algorithms. However, the performance observed here for BDAS



(a) Cost Efficiency (Deadline Factor vs. Cost Ratio)



(b) Time Efficiency (Budget Factor vs. Time Ratio)



(c) Success rate (%) for LIGO workflow with 1000, 5000 and 10000 VMs.

Figure 2.7: Cost efficiency, time efficiency, and success rate (%) of CTTWS vs BDAS for LIGO workflow.

algorithm seems to contrast with what is presented in [81]. This is due to the large range of instance types used in this thesis. The second type of experiment will provide more detailed explanations for that issue.

Even though CTTWS has an average success always better than the one of BDAS, the standard deviation of the success rate (see Table 2.3) of BDAS is always smaller than the one of CTTWS. Therefore, it is not enough to conclude that CTTWS is significantly better than BDAS.

We have conducted statistical analysis to find out if our proposal is significantly more efficient in terms of SR (i.e. customer satisfaction) than BDAS. The Microsoft Excel add-in *Analysis ToolPak*[2] [100] has been used for all our statistical tests. Since here we have two

[2]https://www.excel-easy.com/data-analysis/analysis-toolpak.html

Table 2.3: Success rate summary for the five scientific workflows and the five workloads with 1000, 5000 and 10000 provisioned VMs. CTTWS can have a total average of SR that is up-to 32.5% higher than the one of BDAS

	1000 VMs		5000 VMs		10000 VMs	
Workflow	**CTTWS**	**BDAS**	**CTTWS**	**BDAS**	**CTTWS**	**BDAS**
MONTAGE 50	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
MONTAGE 100	**18.0%**	0.0%	**12.0%**	0.0%	**15.0%**	0.0%
MONTAGE 200	**23.0%**	0.0%	**23.0%**	0.0%	**23.0%**	0.0%
MONTAGE 500	**60.0%**	0.0%	**67.0%**	0.0%	**69.0%**	0.0%
MONTAGE 1000	**89.0%**	0.0%	**25.0%**	0.0%	**25.0%**	0.0%
EPIGENOMICS 50	**75.0%**	14.0%	**74.0%**	22.0%	**73.0%**	20.0%
EPIGENOMICS 100	**77.0%**	21.0%	**77.0%**	29.0%	**75.0%**	27.0%
EPIGENOMICS 200	**73.0%**	30.0%	**71.0%**	28.0%	**69.0%**	35.0%
EPIGENOMICS 500	**73.0%**	3.0%	**69.0%**	46.0%	**64.0%**	44.0%
EPIGENOMICS 1000	**72.0%**	0.0%	**69.0%**	33.0%	**61.0%**	40.0%
CYBERSHAKE 50	**81.0%**	9.0%	**81.0%**	9.0%	**78.0%**	7.0%
CYBERSHAKE 100	**96.0%**	3.0%	**97.0%**	4.0%	**96.0%**	4.0%
CYBERSHAKE 200	**100.0%**	47.0%	**100.0%**	41.0%	**100.0%**	53.0%
CYBERSHAKE 500	**99.0%**	3.0%	**99.0%**	0.0%	**99.0%**	0.0%
CYBERSHAKE 1000	**95.0%**	32.0%	**90.0%**	3.0%	**90.0%**	0.0%
SIPHT 50	0.0%	**31.0%**	0.0%	**34.0%**	0.0%	**33.0%**
SIPHT 100	0.0%	**68.0%**	0.0%	**67.0%**	0.0%	**68.0%**
SIPHT 200	0.0%	**71.0%**	0.0%	**73.0%**	0.0%	**73.0%**
SIPHT 500	25.0%	**62.0%**	0.0%	**63.0%**	0.0%	**63.0%**
SIPHT 1000	**66.0%**	58.0%	0.0%	**46.0%**	0.0%	**51.0%**
LIGO 50	**13.0%**	0.0%	**14.0%**	0.0%	**20.0%**	0.0%
LIGO 100	**21.0%**	0.0%	**25.0%**	0.0%	**25.0%**	0.0%
LIGO 200	**33.0%**	0.0%	**30.0%**	0.0%	**28.0%**	0.0%
LIGO 500	**53.0%**	1.0%	**44.0%**	0.0%	**51.0%**	0.0%
LIGO 1000	**23.0%**	0.0%	**20.0%**	0.0%	**15.0%**	0.0%
Mean	**50.6%**	18.1%	**43.5%**	20.0%	**43.0%**	20.7%
Standard deviation (SD)	35.58%	**24.58%**	36.75%	**24.29%**	36.05%	**25.53%**

algorithms to compare, we have conducted Student's test[3] between the success rate of CTTWS algorithm and BDAS algorithm for the different workflows and the different workloads. We have considered two different hypotheses to compare the proposed algorithm (CTTWS) with BDAS.

- First hypothesis (H_0): There is no difference between the proposed and the comparing algorithm.
- Second hypothesis (H_1) : There is a difference between the proposed and the comparing algorithm.

As commonly accepted [101, 102], we consider a significance level of $p < 0.05$ for our t test (which corresponds to a confidence interval of 95%). According to the provided data in Table

[3]https://www.excel-easy.com/examples/t-test.html

2.4, a notable difference between the proposed algorithm and other baseline algorithm exists since the p-value in all cases is lower than 0.05. The null hypothesis (H_0) is therefore rejected. Thus, this test proves the fact that differences between the proposed CTTWS algorithm and BDAS algorithm is significant in terms success rate. We can then conclude that there is a significant contribution in terms of customer satisfaction, since additionally, CTTWS scored higher average of success rate than BDAS in all situations.

Table 2.4: Student's test result comparing the SR of the proposed CTTWS algorithm with the one of BDAS (for the five scientific workflows and the five workloads with 1000, 5000 and 10000 provisioned VMs)

	1000 VMs		**5000 VMs**		**10000 VMs**	
Algorithm	**CTTWS**	**BDAS**	**CTTWS**	**BDAS**	**CTTWS**	**BDAS**
Mean	**50.6%**	18.1%	**43.5%**	20.0%	**43.0%**	20.7%
SD	35.58%	24.58%	36.75%	24.29%	36.05%	25.53%
t-value	3.755200214		2.664736131		2.526163014	
p-value	**0.000515571**		**0.010884737**		**0.015294934**	

2.4.2 Experiment Type 2

In this experiment type we have considered four different set of instance types among the ones presented in Table 2.2:

1. From *m3.medium* to *m5.8xlarge* ($M3\_M - M5\_8XL$): the six slowest instances;
2. From *m4.4xlarge* to *m5.24xlarge* ($M4\_4XL - M5\_24XL$): the six fastest instances;
3. From *m4.xlarge* to *m5.12xlarge* ($M4\_XL - M5\_12XL$): the six instances of the middle;
4. From *m3.medium* to *m5.24xlarge* ($M3\_M - M5\_24XL$): all the ten instances.

We fixed $\alpha = 16$ and $\beta = 16$, and still conducted 100 experiments for each workflow structure (20 for each of the five sizes). The purpose of these experiments is to determine how well our algorithm is able to evaluate the instance types range in a large list, for good scheduling. In fact, cloud providers often offer a list of different types of instances, which poses the problem of choosing appropriate types of instances leading to a good scheduling.

Performance summary for Experiment type 2

The results of the experiments presented in Figures 2.8a, 2.8c, 2.8b, 2.8d and 2.8e show clearly the impact of the available set of instances over the performance of BDAS and CTTWS algorithms.

When we carefully observe the results of the last set of instances ($M3\_M - M5\_24XL$) for BDAS algorithm, we notice that the success rate obtained is largely smaller that the ones of the three first sets of instances. In the Figure 2.8 it corresponds to the rightmost histogram. The success rate values of BDAS in that rightmost histogram of Figure 2.8 are similar to the ones obtained in the first type of experiments. This means that BDAS has some difficulty to make an adequate choice of VMs when the set of instance types is large and varied.

That is mainly due to the fact that, even though BDAS uses a trade-off function, sometimes because of the subtraction in both cost and time part of this function, we fall into a negative value which produces wrong results, whereas it would still have been possible to find-out a good mapping. This is given by box plots of the first type of experiments, and particularly by the upper outliers observed in most of the cases. However, CTTWS is free of these disadvantages because it uses a finer spare budget evaluation for the cost part, and the inverse of the finish time for the time part.

Another explanation is the usage of the IRITR in the CTTWS algorithm which has the ability to reduce the set of instance type to a suitable range of instance types. In fact, since the last set of instance types contains all the instance types, it become difficult for BDAS to chose good instance. Whereas, in the experiments with a small set of instance (which correspond to the three set of instance of in Table 2.5) BDAS has good success rate.

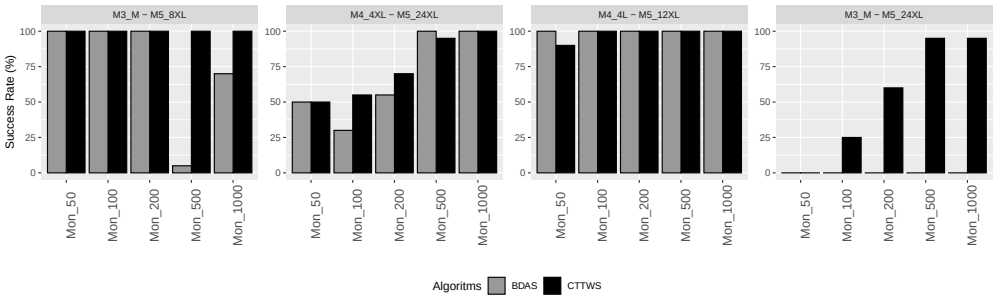
The results of these experiment prove the effectiveness of our algorithm as summarized in Table 2.5. In particular, it highlights the importance of the Implicit Requested Instance Types Range (IRITR) evaluation in a scheduling plan.

Table 2.5: Success rate summary for the five scientific workflows and the different set of instance types. CTTWS can have a total average of SR that is up-to 38.4% higher than the one of BDAS according to the available set of instance types.

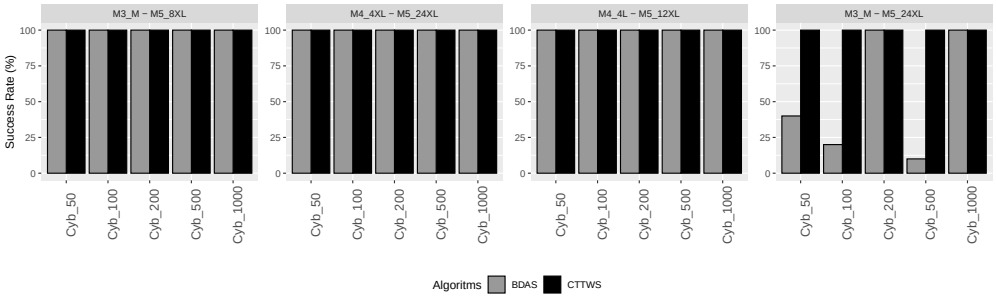
	M3_M - M5_8XL		***M4_4XL - M5_24XL***		***M4_XL - M5_12XL***		***M3_M - M5_24XL***	
Workflow	**CTTWS**	**BDAS**	**CTTWS**	**BDAS**	**CTTWS**	**BDAS**	**CTTWS**	**BDAS**
MONTAGE	100%	75%	74%	67%	98%	100%	55%	0%
CYBERSHAKE	100%	100%	100%	100%	100%	100%	100%	54%
EPIGENOMICS	100%	100%	100%	100%	100%	100%	100%	28%
SIPHT	100%	100%	100%	100%	100%	100%	40%	87%
LIGO	85%	80%	85%	94%	85%	90%	66%	0%
Mean	**97.00%**	91.00%	91.80%	**92.20%**	96.60%	**98.00%**	**72.20%**	33.80%

2.5 Quality improvement analysis of the CTTWS algorithm

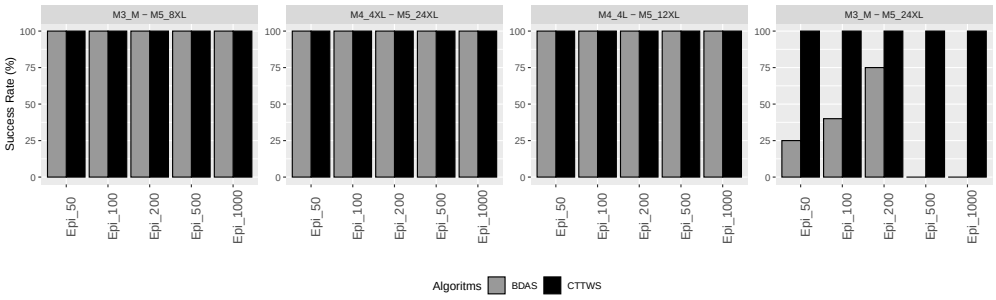
The conceptual differences between CTTWS and BDAS can be summarized in four points. First, the task selection strategy used in CTTWS is based on a fine-grained approach, and as a



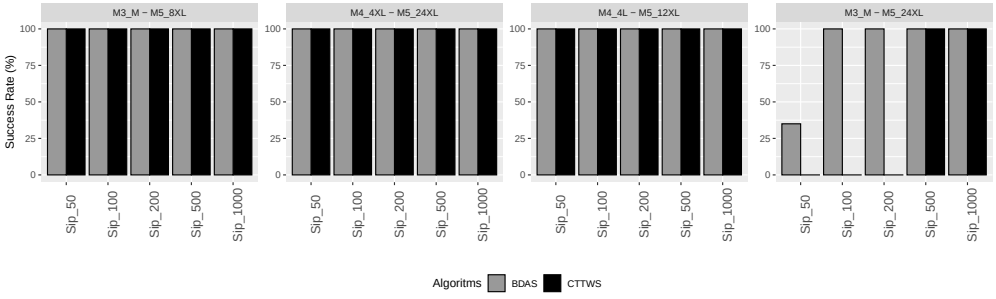
(a) Success rate (%) for MONTAGE Workflow with different set of instance types.



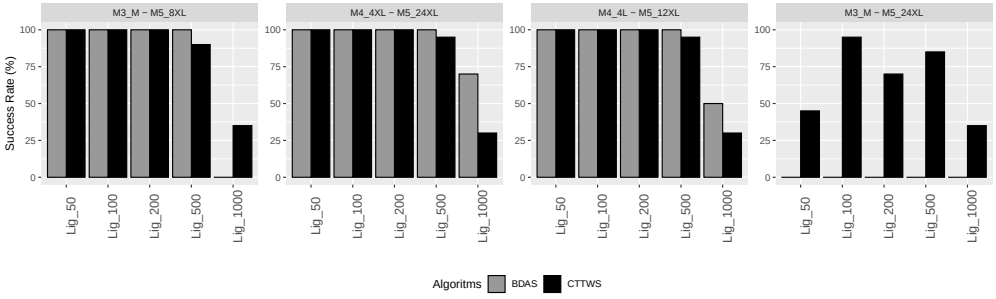
(b) Success rate (%) for CYBERSHAKE Workflow with different set of instance types.



(c) Success rate (%) for EPIGENOMICS Workflow with different set of instance types.



(d) Success rate (%) for SIPHT Workflow with different set of instance types.



(e) Success rate (%) for LIGO Workflow with different set of instance types.

Figure 2.8: Success rate (%) of CTTWS vs BDAS for five workflows with different set of instance types.

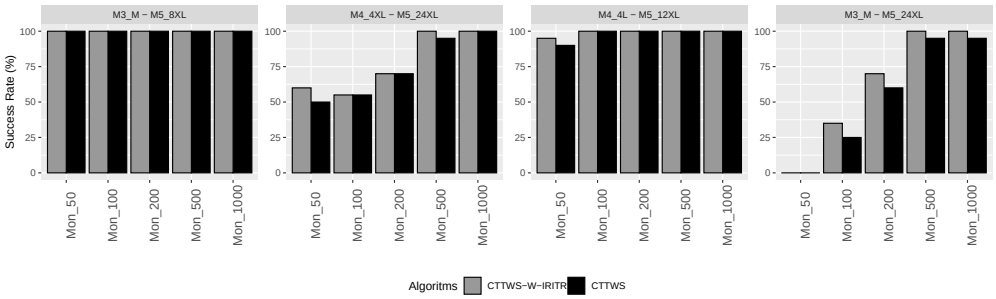
consequence, a task is ready for execution when its parents have been executed, while the one used in BDAS is based on a big-grained approach, and as a consequence, it may force a task to wait on non-dependencies. Second, the spare budget management strategy used in CTTWS is based on fine-grained updates while the one used in BDAS is based on big-grained updates. Third, the trade-off function used in CTTWS is based on the weight of the current task, its added cost, and the current spare budget, while the one used in BDAS is based on the weight of the current task and the deadline and the budget of its level. Thus, the CTTWS trade-off function is based on a fine-grained approach as only one task is used, the current task, while its BDAS counterpart is based on a big-grained approach as it involves a task level. Fourth, the new scheduling tool introduced in the CTTWS algorithm, the IRITR evaluation.

In order to highlight the level of relevance of the conceptual differences, we conducted a sensitivity analysis. To this end, we remove the IRITR evaluation from the CTTWS algorithm, which corresponds to ignoring lines 9 and 10 of the algorithm 1 and line 2 of the algorithm 2. We conducted the experiments of type 2 on the resulting algorithm, named CTTWS without IRITR and denoted by CTTWS-W-IRITR.

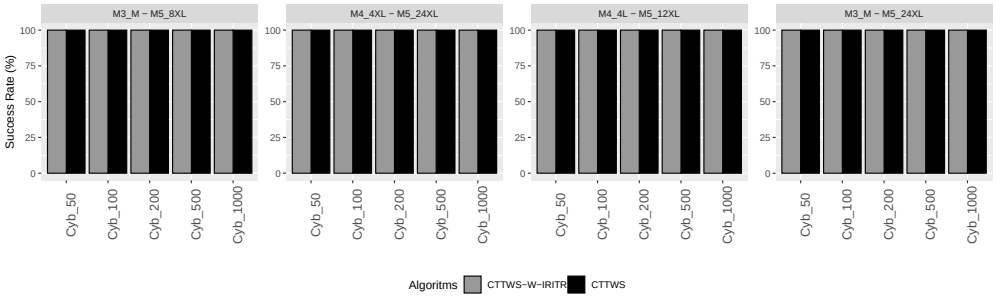
The results of the experiments summarized in Table 2.6 show clearly the impact of the IRITR in the scheduling when the set of instance types is large. CTTWS outperforms CTTWS-W-IRITR up-to 10.2% of SR average for the larger set of instance types $M3\_M - M5\_24XL$. CTTWS and CTTWS-W-IRITR score respectively 89.40% and 87.20% of total SR average.

However, when we carefully observe the detailed results of Figures 2.9a, 2.9c, 2.9b, 2.9d and 2.9e, we realize that the type and the size of the workflow have an impact over the effectiveness of CTTWS, since CTTWS-W-IRITR sometimes gets the better SR. Thus, It would have been better not to use the IRITR evaluation in some situations.

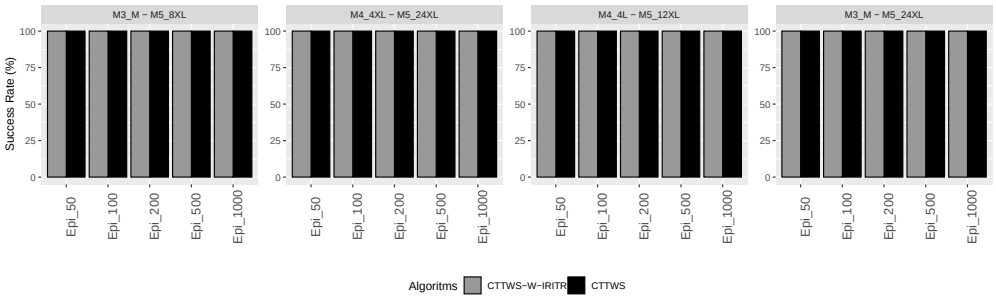
For the conceptual aspects of the trade-off function and its related issues, task selection, and budget management, we need only to compare the results obtained by BDAS and CTTWS-W-IRITR for the experiments of type 2, presented in Tables 2.5 and 2.6. BDAS and CTTWS-W-IRITR score respectively 78.75% and 87.20% of the total SR average, meaning that CTTWS-W-IRITR outperforms BDAS by 8.45%. Thus, exploiting a fine granularity approach in the trade-off function and its associated issues gives better results than the big granularity approach.



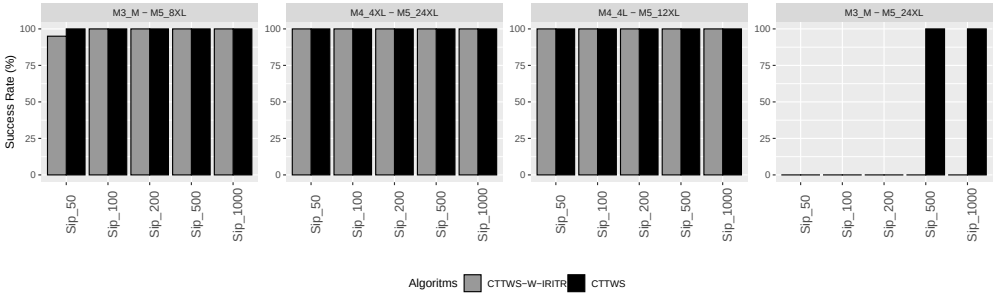
(a) Sensitivity of the IRITR over the SR (%) for MONTAGE Workflow.



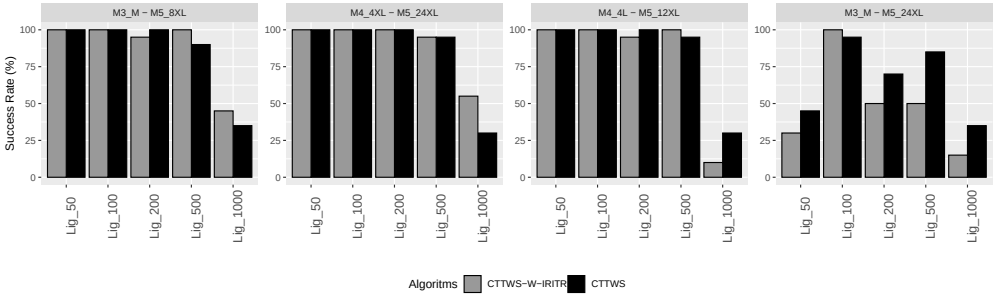
(b) Sensitivity of the IRITR over the SR (%) for CYBERSHAKE Workflow.



(c) Sensitivity of the IRITR over the SR (%) for EPIGENOMICS Workflow.



(d) Sensitivity of the IRITR over the SR (%) for SIPHT Workflow.



(e) Sensitivity of the IRITR over the SR (%) for LIGO Workflow.

Figure 2.9: Success rate (%) of CTTWS vs CTTWS-W-IRITR for the five workflows with different set of instance types.

Table 2.6: Success rate summary for the five scientific workflows and the different set of instance types. CTTWS can have a total average of SR that is up-to 10.2% higher than the one of CTTWS-W-IRITR according to the available set of instance types.

	M3_M - M5_8XL		***M4_4XL - M5_24XL***		***M4_XL - M5_12XL***		***M3_M - M5_24XL***	
Workflow	**CTTWS**	CTTWS-W-IRITR	**CTTWS**	CTTWS-W-IRITR	**CTTWS**	CTTWS-W-IRITR	**CTTWS**	CTTWS-W-IRITR
MONTAGE	100%	100%	74%	77%	98%	99%	55%	61%
CYBERSHAKE	100%	100%	100%	100%	100%	100%	100%	100%
EPIGENOMICS	100%	100%	100%	100%	100%	100%	100%	100%
SIPHT	100%	99%	100%	100%	100%	100%	40%	0%
LIGO	85%	88%	85%	90%	85%	81%	66%	49%
Mean	97.00%	97.40%	91.80%	93.40%	96.60%	96.00%	72.20%	62.00%

Conclusion

In this chapter, we have presented our first proposal, the Cost-Time Trade-off efficient Workflow Scheduling (CTTWS) algorithm in commercial cloud environments. The CTTWS algorithm aims at minimizing both execution cost and execution time, in the respect of user-defined budget and deadline.

Intensives experiments have been conducted considering five type workflows and five workload, associated to twelve combinations of deadline and budget. The results of the experiments supported by a Student's T-Test, proved that CTTWS can produce better success rates to meet users' deadlines and budgets up-to 38.4% higher than the one of BDAS according to the variety of available instance types. CTTWS is more effective when the range of instance types is increasing. Thus, paying attention to the types of resources, their number, and the granularity of the set of tasks exploited in the different scheduling steps is very important. In addition, CTTWS is simple, with low complexity and overhead.

The innovations brought by the CTTWS, which have been highlighted as the main assets of the algorithm are the IRITR evaluation, the fine-grained spare budget evaluation, and the fined-grained VM selection.

However, the CTTWS algorithm considers static VMs provisioning, while resources are actually dynamically provisioned in Cloud environments. In addition to that, we have realized that the algorithm has poor performance when it comes to the SIPHT workflow. This is mainly due to the great number of tasks at the entry of the workflow. That means that the number of VMs has not been handle appropriately.

CHAPTER

3 Dynamic provisioning based workflow scheduling with budget and deadline awareness

Since execution cost minimization and completion time minimization are contradictory objectives, addressing such issue through trade-off function approaches have proved to be an efficient way. However, our Cost-Time Trade-off efficient Workflow Scheduling (CTTWS) algorithm presented in the chapter 2, though efficient, has some shortcomings. Namely the Static provisioning and the inability to perform effectively notwithstanding the types of workflow. Therefore, in this chapter, we proposed another algorithm called: Cost-Time Trade-off efficient Workflow Scheduling with Dynamic provisioning (CTTWSDP). The CTTWSDP algorithm relies on a dynamic VMs provisioning with a limited number of leased VMs. In addition, an improved Implicit Requested Instance Types Range (IRITR) evaluation is proposed. The results of simulations prove the effectiveness of the proposal en highlight the level of improvement of CTTWSDP over CTTWS. Moreover, our algorithm achieves at overall a 17.09% – 44.80% higher success rate when compared to three state-of-the-art algorithms. Furthermore, ANOVA test along with pairwise tests using Tukey-Kramer have been conducted, revealing that CTTWSDP is significantly more effective than two of the baseline algorithm, while for the third one the out-performance of CTTWSDP is not statistically significant. An analysis of the standard deviation of the success rate proves that CTTWSDP is more stable in its performance no matter the type and the workload of workflow. With a standard deviation of 2.42, smaller than those of other algorithms from 13.35 to 35.69.

Introduction

We have proposed in the previous chapter the Cost-Time Trade-off efficient Workflow Scheduling (CTTWS) algorithm which produces effective result compared to a state-of-the-art algorithm. However, some ways of improvement have been unveiled. their are the lack of dynamicity in the provisioning process, and the inefficiency when it come to some type of workflow.

In this chapter, we proposed an improved version of the CTTWS algorithm called: Cost-

This chapter is derived from: **J. E. Ndamlabin Mboula**, V. C. Kamla, and C. Tayou Djamegni: *Dynamic provisioning with structure inspired selection and limitation of VMs based cost-time efficient workflow scheduling in the cloud.* Cluster Computing 24(2021): 2697-2721, Springer.

Time Trade-off efficient Workflow Scheduling with Dynamic provisioning (CTTWSDP). In addition to the dynamic VMs provisioning that is limited in number of leased VMs, the CT-TWSDP algorithm improves the Implicit Requested Instance Types Range (IRITR) evaluation.

The proposal is compared to three state-of-the-art algorithms. Namely, the Deadline Constrained Critical Path (DCCP) algorithm [87], the Partition Problem based Dynamic Provisioning and Scheduling (PPDPS) algorithm [45] and the Greedy Resource Provisioning and modified HEFT (GRP-HEFT) [44]. Comparative simulation results proved the effectiveness of our proposal.

The remaining of this chapter is as follows. We first present the modeling of the workflow scheduling problem. Secondly, we present the details of the proposed algorithm. Thirdly, we describe the performance evaluation. Fourthly, we analyse the simulation results, and finally, we end the chapter by a conclusion.

3.1 Modeling of the workflow scheduling problem

In this section we present the cloud resource model, the workflow model, and the problem formulation.

3.1.1 Cloud computing model

The main difference between the cloud model considered in this chapter and the one of chapter 2 lies in the resource provisioning strategy. Here we assume that the VMs are dynamically provisioned according to the need, and can be of any of the instance types. We assume that at any moment a number of P VMs are available (VMS $= \{vm_1, vm_2, ..., , vm_p, ..., vm_P\}$). The VMs are leased as subscription-based in a pay-per-use model and are charged per billing period of length τ. A billing period is one hour per VM usage for most IaaS providers. The dynamic provisioning involves taking into account the creation and launching time of VMs which is called provisioning delay.

Apart from the above-mentioned difference, all the aspects presented in section 2.1.1 remain unchanged.

3.1.2 Workflow model

The workflow model considered in this chapter is the same as that of chapter 2. The description of a scientific workflow represented as a DAG is given in section 2.1.2.

3.1.3 Problem formulation

In this chapter, the assumption is that the VMs are dynamically provisioned as scheduling progress according to the need. A workflow scheduling algorithm aims at determining an execution order of the workflow tasks, and the VM onto which to assign each task. The execution order of workflow tasks and the mapping of tasks onto VMs must be done in order to respect the user requirements, the workflow constraints as well as the provider requirement. The user requirements are the user-defined deadline δ and the user-defined budget B. The workflow constraints are made of task weights and data dependencies between them; that is, a task cannot start its execution before its parents and the reception of the data needed for its execution. While the provider requirement considered here is the maximization of resources (VMs) usage, that means using less possible VMs to satisfy the user.

The question to deal with is: *how to build a workflow scheduling algorithm able to dynamically provision VMs for tasks execution in order to reduce the overall execution cost and execution time in the respect of the user-defined budget and deadline?*

The problem can be formulated as a mathematical optimization problem:

$$\begin{cases} Reduce(M_G) \\ Reduce(Cost_G) \\ Subject\ to\ M_G \leq \delta\ and\ Cost_G \leq B \end{cases} \tag{3.1}$$

3.2 The Proposed Scheduling Algorithm

In this section, we present our proposed solution for the workflow scheduling problem, the Cost-Time Trade-off efficient Workflow Scheduling with Dynamic provisioning (CTTWSDP), which aims at optimizing both processing costs and times. The CTTWSDP scheduling algorithm maintains the four main steps of the CTTWS algorithm that are summarized in Table 2.1. Namely Spare Budget Evaluation, (Improvement) Implicit Requested Instance Types Range (IRITR) evaluation, Task Selection and VM Selection.

3.2.1 Spare Budget Evaluation

Starting spare budget is equal to the user-defined budget, the spare budget is recomputed after the mapping of each task of the workflow. The execution cost incurred by the schedule of the task is deducted from the spare. The principle is the same as in the case of the CTTWS algorithm.

3.2.2 Improvement of the Implicit Requested Instance Types Range (IRITR) evaluation

The computing power of different types of resources is generally proportional to their price, as in Amazon EC2 [96]. As users and cloud providers know, the cost of processing plays a very important role. It is based on the billing model, the planning algorithm and the workflow structure. In an unknown and diverse market, if we do not have a good sales consultant, it will be difficult to optimize the quality of purchases against the budget. In a cloud, this problem of optimizing the budget-quality ratio becomes the problem of optimizing both cost and makespan with respect to both budget and deadline. This last problem is solved here by avoiding both overbidding and underbidding of resources during the execution of the workflow. To solve this avoidance problem, we introduce a new metric, the Implicit Requested Instance Types Range (IRITR), which represents a range of instance types that the budget can afford based on makespan, deadline and the workflow structure.

In this chapter, we propose an improved version of the IRITR evaluation. Apart from the usage of a dynamic provisioning strategy against the static one used in our previous chapter, the improvement of the IRITR evaluation constitutes a major difference.

The improved IRITR construction process is described in the following.

- The Minimum and Maximum Remaining Number of Billing Period ($MinRNBP$, $MaxRNBP$) are defined respectively by

$$MinRNBP = \left\lceil \frac{remM_G}{\tau} \right\rceil \times \mathbf{AvgW_{Wf}}; \tag{3.2}$$

and

$$MaxRNBP = \left\lceil \frac{rem\delta_G}{\tau} \right\rceil \times \mathbf{AvgW_{Wf}}; \tag{3.3}$$

- The Minimum and Maximum Implicit Requested Instance Type Cost ($MinIRITC$, $MaxIRITC$) are defined respectively by

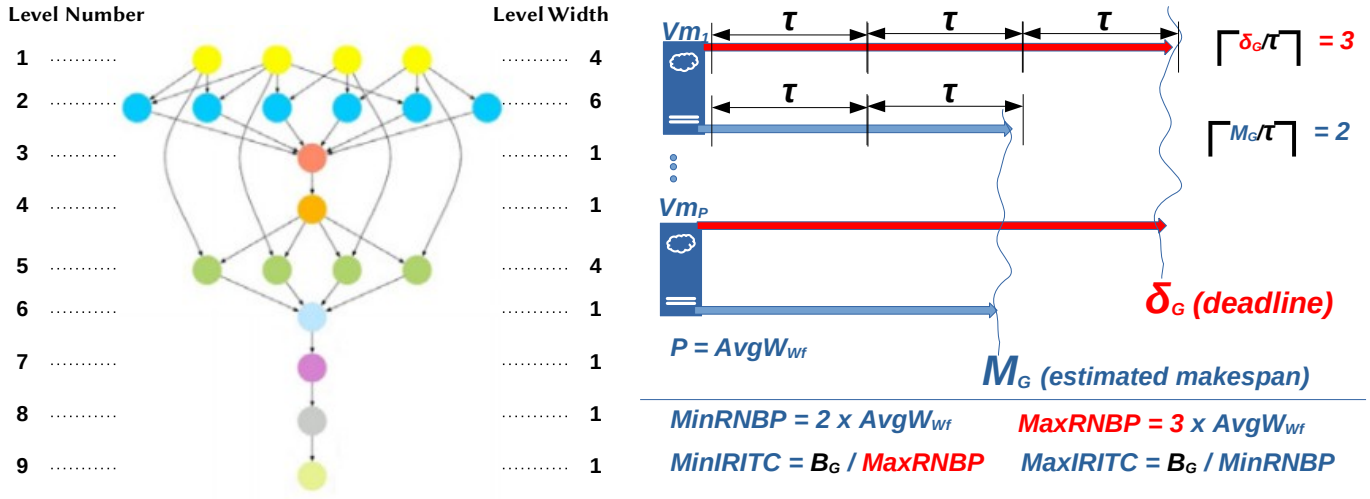
$$MinIRITC = \frac{remB_G}{MaxRNBP}; \tag{3.4}$$

and

$$MaxIRITC = \frac{remB_G}{MinRNBP} \tag{3.5}$$

where $remM_G$ is the remaining makespan, $rem\delta_G$ the remaining deadline, δ the deadline and $remB_G$ the remaining budget, and $AvgW_{Wf}$ the average width of the workflow which is described in the following.

Figure 3.1b presents an illustrative explanation of the improved IRITR evaluation.



(a) Workflow widths distribution ($AvgW_{Wf} = 2$)

(b) Improved IRITR explanation

Figure 3.1: Improved IRITR illustration

Figure 3.1a shows the structure of a Montage workflow with twenty tasks and their dependencies. In this figure, the left column shows level numbers calculated by equation (3.6), while the right column is the number of tasks in each level that we call the level width ($levelWidth(l)$). In the example of Figure 3.1a, the largest width is 6 which corresponds to the level 2 ($levelWidth(2) = 6$), and the average width of the workflow $AvgW_{Wf} = 2$.

$$LN(t_j) = 1 + \max_{t_i \in pred(t_j)} \{LN(t_i)\}, \tag{3.6}$$

From equations (3.2), (3.3), (3.4) and (3.5), we determine the slowest ($cheapestIRIT$) and the fastest ($expensiveIRIT$) VMs instance types whose the costs are between $MinIRITC$ and $MaxIRITC$, and we set the range as $IRITR = \{vmit_k | MinIRITC \leq c_k \leq MaxIRITC\} = [cheapestIRIT, expensiveIRIT]$.

During the execution of a workflow, the remaining makespan and deadline at a any time t are respectively $remM_G = M_G - t$ and $rem\delta_G = \delta - t$. Since the makespan is unknown during the execution of the workflow, it is approximated by $minM_G$ and the remaining makespan is approximated as follows: $remMG \simeq minM_G - t$. The $IRITR$ could be constructed either at any time a task has to be mapped onto a VM (also called VM selection) or only at the beginning of the mapping process. In order to minimize the load of the mapping process, we choose the second. In this case, $remM_G \simeq minMG - AST(t_{first}) \simeq minMG$ and $rem\delta_G =$

$\delta - AST(t_{first}) = \delta_G$ as $AST(t_{first}) = 0$.

The same robustness management is considered here, as in chapter 2.

The main difference between the improved IRITR evaluation presented here and the one in chapter 2 lies on the usage of average width, "$AvgW_{Wf}$", in the formulas of the improved version (see equations (3.2) and (3.3)) instead of the number of unmapped tasks , "$remNbTasks$", as it is done in chapter 2.

3.2.3 Task Selection

Tasks are ordered in a list, called ready list, according to their Earliest Start Time (EST) so that all the root tasks receive the highest priority. The Earliest Start Time has been defined in section 2.1.2 by the equation (2.7).

3.2.4 VM Selection

Once the workflow tasks have been ordered according to their EST, they are ready to be mapped onto the available VMs, one after the other. The trade-off between cost and time is handled according to the trade-off function described for the CTTWS algorithm in the section 2.2.4.

Dynamic VM Provisioning algorithm

It seems unsuitable to use more VMs than the largest width of the workflow for its execution. But at the same time how many VMs is suitable for the execution of the workflow? According to the level of parallelism of the workflow due to its structure, we use the average width $AvgW_{Wf}$ (described in section 3.2.2) of the workflow as limit of the number of VMs to provision for the execution.

A new VM is provisioned in preference to the best among the available VMs, if both of the following conditions are met:

- the ready time of the best VM is greater than the Min ready time of the current task (bestReadyTime > minReadyTime$_i$);
- the number of already provisioned VMs < $AvgW_{Wf}$.

The pseudo-code of the dynamic VM Provisioning algorithm is depicted in Algorithm 3.

Algorithm 3 Dynamic VM Provisioning Algorithm

Input: t_i

Output: t_i is mapped to the suitable VM ($bestVM$)

if t_i **has already been mapped to a VM then**
    return;/*do nothing since this task has been mapped through one of its parents. It is the case of tasks in the same pipeline. */
end if
for $vm_p \in VMS$ **do**
    Calculate the $CTTF_i^p$
    $bestVM \leftarrow vm_p$ **if** vm_p **fulfils the following conditions:**
    (C1) vm_p **have the highest** $CTTF_i^p$
    (C2_1) If t_i **is a root task, the VM** vm_p **must be of instance type** $expensiveIRIT$
    (C2_2) If t_i **is not a root task, the VM** vm_p **must be of an instance type belonging to** $IRITR$
end for
/*dynamic provisioning*/
if $bestReadyTime > minReadyTime_i$ **and** $length(VMS) < optNbVMs$ **then**
    for $vmit_k \in VMIT$ **do**
        Calculate the $CTTF_i^k$
        $bestVMInstance \leftarrow vmit_k$ **if** $vmit_k$ **fulfils the following conditions:**
        (C1) $vmit_k$ **have the highest** $CTTF_i^k$
        (C2_1) If t_i **is a root task,** $vmit_k$ **must be equal to** $expensiveIRIT$
        (C2_2) If t_i **is not a root task,** $vmit_k$ **must belong to** $IRITR$
    end for
    $bestVM \leftarrow vmProvisioning(bestVMInstance)$
end if
Map t_i **to** $bestVM$
$t_{parent} \leftarrow t_i$
$t_{next} \leftarrow child(t_{parent})[0]$
while t_{parent} **has exactly one child &&** t_{next} **has exactly one parent &&** t_{next} **adds zero cost on** $bestVM$ **do**
    Map t_{next} **to** $bestVM$
    $t_{parent} \leftarrow t_{next}$
    $t_{next} \leftarrow child(t_{next})[0]$
end while
$spareB^t \leftarrow spareB^t - EC(i,p)_{added}$

3.2.5 The CTTWSDP algorithms

Here, we present the Cost-Time Trade-off efficient Workflow Scheduling with Dynamic provisioning (CTTWSDP) algorithm. The CTTWSDP algorithm consists of four main steps as mentioned above and strives to enable the cloud scheduler to spend less money to complete a workflow without exceeding the deadline.

The pseudo-code of the CTTWSDP algorithm is depicted in Algorithm 4.

3.2.6 An illustrative example

To illustrate how our proposed CTTWSDP algorithm works, we consider the workflow of Figure 3.2 and the resources of Table 2.2. We also consider the parameters of Table 3.1.

Algorithm 4 CTTWSDP Algorithm

Input: The DAG of tasks

Output: All the tasks are scheduled to their suitable VMs

1: **Order the tasks** $readyList$ **in Asc** EST
2: $[cheapestIRIT, expensiveIRIT] \leftarrow IRITR$ **Evaluation**
3: **for** $t_i \in readyList$ **do**
4: **Call the Dynamic VM Provisioning Algorithm on** t_i
5: **Schedule task** t_i **to its mapped VM** vm_p
6: **end for**

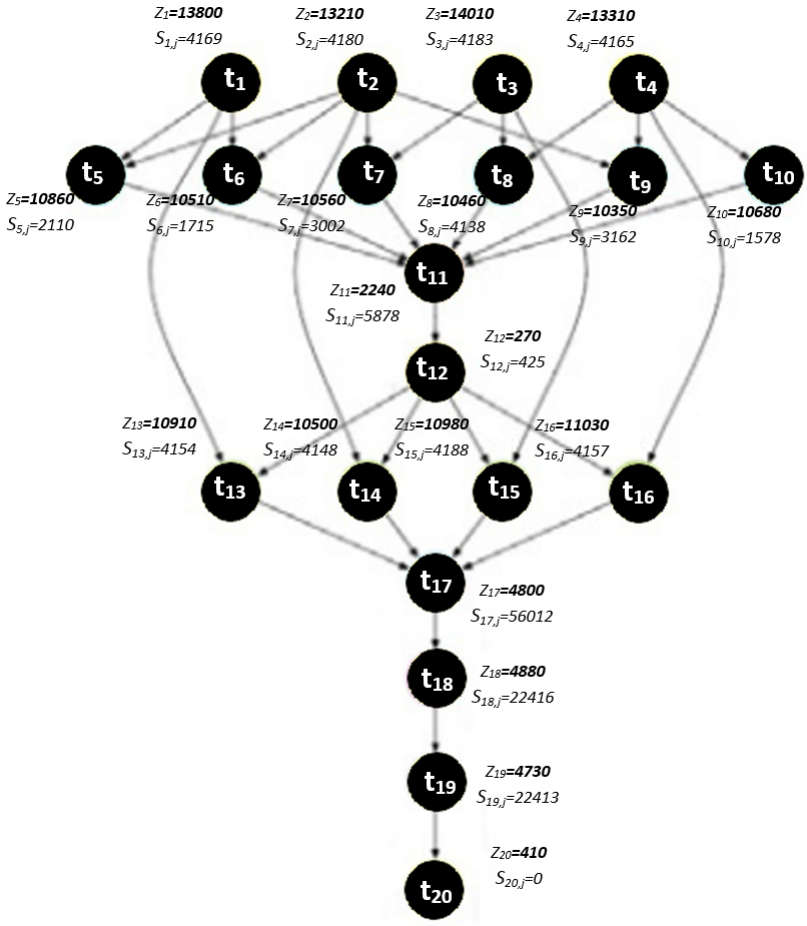


Figure 3.2: Illustrative example: Montage workflow with 20 tasks (***Z_i is the length of the task t_i in MI, and $s_{i,j}$ the size of the data (in MB) from t_i to t_j***.

Table 3.1: CTTWSDP algorithm illustrative example: parameters

DF	**BF**	**FS (in sec.)**	**LB (in \$)**	δ_G **(in sec.)**	B_G **(in \$)**
16	4	847.71	1.541	13552	6.16

IRITR evaluation

Let us evaluate the $MinRNBP$ and the $MaxRNBP$:

$$\begin{aligned} MinRNBP &= \left\lceil \frac{remM_G}{\tau} \right\rceil \times AvgW_{Wf}, \\ &= \left\lceil \frac{minM_G}{\tau} \right\rceil \times 2, \\ &= \left\lceil \frac{EFT(t_{20})}{\tau} \right\rceil \times 2, \\ &= \left\lceil \frac{6309}{3600} \right\rceil \times 2, \\ &= 4 \end{aligned} \qquad \begin{aligned} MaxRNBP &= \left\lceil \frac{rem\delta_G}{\tau} \right\rceil \times AvgW_{Wf}, \\ &= \left\lceil \frac{\delta_G}{\tau} \right\rceil \times 2, \\ &= \left\lceil \frac{13552}{3600} \right\rceil \times 2, \\ &= 8 \end{aligned}$$

Therefore, the $MinIRITC$ and the $MaxIRITC$ are:

$$\begin{aligned} MinIRITC &= \frac{remB_G}{MaxRNBP}, \\ &= \frac{B_G}{MaxRNBP}, \\ &= \frac{6.16}{8}, \\ &= 0.77 \end{aligned} \quad and \quad \begin{aligned} MaxIRITC &= \frac{remB_G}{MinRNBP}, \\ &= \frac{B_G}{MinRNBP}, \\ &= \frac{6.16}{4}, \\ &= 1.54 \end{aligned}$$

We then have $IRITR = \{vmit_5, vmit_6\}$ (ie. m4.4xlarge and m5.8xlarge from Table 2.2).

Tasks prioritization

The workflow tasks are ordered in a list, called ready list, according to their Earliest Start Time (EST). Table 3.2 present the priority determination between the tasks. The ordered list is therefore $orderedList = \{t_1, t_2, t_3, t_4, t_7, t_8, t_5, t_6, t_9, t_{10}, t_{11}, t_{12}, t_{13}, t_{14}, t_{15}, t_{16}, t_{17}, t_{18}, t_{19}, t_{20}\}$

VM selection for each task

Ones the tasks have been ordered and the IRITR evaluated, we can proceed to the VM selection. Table 3.3 presents the WM selection process for the twenty tasks. The tasks t_1, t_2, t_3, and t_4 are ready at time zero because they are root tasks. As root task, t_1 is mapped onto the fastest instance of the $IRITR$, which is $vmit_6$. A VM (vm_1^6) is therefore provisioned. As for t_2, due to its min ready time (zero), it can't wait the end of t_1. A new instance of type $vmit_6$ (vm_2^6) is created for its execution. Since the limit of the number of VMs is reached, no more provisioning will be possible (See Algorithm 3. The condition of line 11 will always be false.).

Table 3.2: CTTWSDP algorithm illustrative example: tasks prioritization

Levels	**Priorities**						
1	t_i EST(t_i) Pr(t_i)	t_1 0 1	t_2 0 2	t_3 0 3	t_4 0 4		
2	t_i EST(t_i) Pr(t_i)	t_5 352.2 7	t_6 352.2 8	t_7 358.09 5	t_8 358.09 6	t_9 346.89 9	t_{10} 346.89 10
3, 4	t_i EST(t_i) Pr(t_i)	t_{11} 670.95 11	t_{12} 988.18 12				
5	t_i EST(t_i) Pr(t_i)	t_{13} 1012.24 13	t_{14} 1012.24 14	t_{15} 1012.24 15	t_{16} 1012.24 16		
6, 7, 8, 9	t_i EST(t_i) Pr(t_i)	t_{17} 1334.98 17	t_{18} 4185.58 18	t_{19} 5357.21 19	t_{20} 6527.13 20		

The VM selection steps will rely on the evaluation of the $CTTF$. The cost part in this case will always be the same for all the VMs, because the added cost is always zero. Therefore, it is the early AFT of the task that will be determinant. Apart from the cases of pipelines, the VM with the highest $CTTF$ is determined for the current task. In the cases of the pipeline, the parent task is used to map the rest of the tasks in the pipeline (See Algorithm 3, lines 22 to 28). We have for example $\{\mathbf{t_{11}}, t_{12}\}$, and $\{\mathbf{t_{17}}, t_{18}, t_{19}, t_{20}\}$.

Table 3.3: CTTWSDP algorithm illustrative example: VM selection for each task

Levels	**VM selection process according to Algorithm 3**										
	t_i	$minReadyT(t_i)$	$AST(t_i)$	$spareB^t$ (in \$)	$ET(t_i)$	$AFT(t_i)$	$EC(i,p)_{added}$	$CTTF_i^p$ eval. & VM selection	Selected VM Type	Selected VM	Pipeline treatment
1	t_1	0	0	6.16	431.25	431.25	1.536	Cond. of line 16: is true. (C2_1)	$vmit_6$	vm_1^6	–
	t_2	0	0	5.35	412.81	412.81	1.536	Cond. of line 11: is true.	$vmit_6$	vm_2^6	–
	t_3	0	412.81	3.088	437.81	850.62	0	vm_2^6 has the highest $CTTF$	–	vm_2^6	–
	t_4	0	431.25	3.088	415.94	847.19	0	vm_1^6 has the highest $CTTF$	–	vm_1^6	–
2	t_7	850.62	850.62	3.088	330	1180.62	0	vm_2^6 has the highest $CTTF$	–	vm_2^6	–
	t_8	1059.77	1059.77	3.088	326.87	1386.64	0	vm_1^6 has the highest $CTTF$	–	vm_1^6	–
	t_5	639.7	1180.62	3.088	339.37	1519.99	0	vm_2^6 has the highest $CTTF$	–	vm_2^6	–
	t_6	639.7	1386.64	3.088	328.44	1715.08	0	vm_1^6 has the highest $CTTF$	–	vm_1^6	–
	t_9	1055.44	1519.99	3.088	323.44	1843.43	0	vm_2^6 has the highest $CTTF$	–	vm_2^6	–
	t_{10}	1055.44	1715.08	3.088	333.75	2047.83	0	vm_1^6 has the highest $CTTF$	–	vm_1^6	–
3, 4	t_{11}	2047.83	2047.83	3.088	70	2117.83	0	vm_1^6 has the highest $CTTF$	–	vm_1^6	t_{12}
	t_{12}	2117.83	2117.83	3.088	8.44	2126.27	0	mapped in t_{11}	–	vm_1^6	–
5	t_{13}	2126.27	2126.27	3.088	340.94	2467.19	0	vm_1^6 has the highest $CTTF$	–	vm_1^6	–
	t_{14}	2147.52	2147.52	3.088	328.12	2475.64	0	vm_2^6 has the highest $CTTF$	–	vm_2^6	–
	t_{15}	2467.19	2467.19	3.088	343.12	2810.31	0	vm_1^6 has the highest $CTTF$	–	vm_1^6 *(End)*	–
	t_{16}	2475.64	2475.64	3.088	344.69	2820.33	0	vm_2^6 has the highest $CTTF$	–	vm_2^6	–
6, 7, 8, 9	t_{17}	2820.33	3019.71	3.088	150	3169.71	0	vm_2^6 has the highest $CTTF$	–	vm_2^6	t_{18}, t_{19}, t_{20}
	t_{18}	3169.71	3169.71	3.088	152.5	3322.22	0	mapped in t_{17}	–	vm_2^6	–
	t_{19}	3322.22	3322.22	3.088	147.81	3470.03	0	mapped in t_{17}	–	vm_2^6	–
	t_{20}	3470.03	3470.03	**3.088**	12.81	**3482.84**	0	mapped in t_{17}	–	vm_2^6 *(End)*	–

3.2.7 Time Complexity

For the same raisin than in the case of CTTWS, to determine the time complexity of CTTWSDP algorithm, the phases that must be considered are: task selection and VM selection. For the task selection, given a workflow containing n tasks, we need $O(n^2)$ time for the determination of their EST. Sorting the tasks takes $O(nlogn)$ time complexity, which gives an overall of $O(n^2)$ for the task selection. For each ready task, to select the suitable VM all the VMs should be examined. Which gives a time complexity of $O(n \times P)$ for VM selection, where P is the number of VMs. Which gives a time complexity of $O(n \times (n + P))$.

However, since the number of VMs is limited to the average width of the workflow, we have $P < n$. Therefore CTTWSDP algorithm has a polynomial time complexity of $O(n^2)$.

The complexity of the other algorithms, DCCP [87], PPDPS [45], and GRP-HEFT [44] have been determined and presented in the related publications. While PPDPS has a time complexity of $O(n^2)$, DCCP and GRP-HEFT have respectively $O(n^2 \times P)$ and $O(n^2 \times K)$. K is the number of used instance types and P the number of provisioned VMs instances, with $K \leq P$ and $P \leq n$. While K is significantly small, P grows significantly with n. For instance, in the case of Montage_1000 the DCCP algorithm effectively uses 518 VMs, that is almost $2 \times n$. Therefore, we can conclude that DCCP and GRP-HEFT have respectively $O(n^3)$ and $O(n^2)$ time complexity order.

A complexity comparison between the studied algorithms is presented in Table 3.4.

Table 3.4: Complexity comparison. *Where n is the number of tasks of the workflow, K the number of used instance types and P the number of provisioned VMs instances.*

Algorithms	**DCCP**	**PPDPS**	**GRP-HEFT**	**CTTWSDP**
Complexity	$O(n^3)$	$O(n^2)$	$O(n^2)$	$O(n^2)$

3.3 Performance evaluation

For the simulations we consider the system as a single datacenter having ten different instance types that are based on the US-east (Ohio) Amazon region [95], collected in July 2019, and which the characteristics are presented in Table 2.2 (see section 2.3).

We set-up the simulation environment as follows. The bandwidth between instances is fixed to 20 MBps, the value of the vCPU of each instance is considered as its processing capacity in Million Instruction Per Second (MIPS) as seen in [41]. The charging model has being configured to reflect the Amazon EC2 instances charge that is an hourly interval from the time of provisioning. Since the algorithms use the dynamic provisioning of VMs, the provisioning

delay of each VM was set to 100 s based on the study by Mao et *al.* [103].

3.3.1 Performance Metrics

We compare the performances of the different algorithms based on the following well-known performance metrics that are already define in chapter 2 and section 2.3.2: Cost Ratio (CR), Time Ratio (TR) and Success Rate (SR).

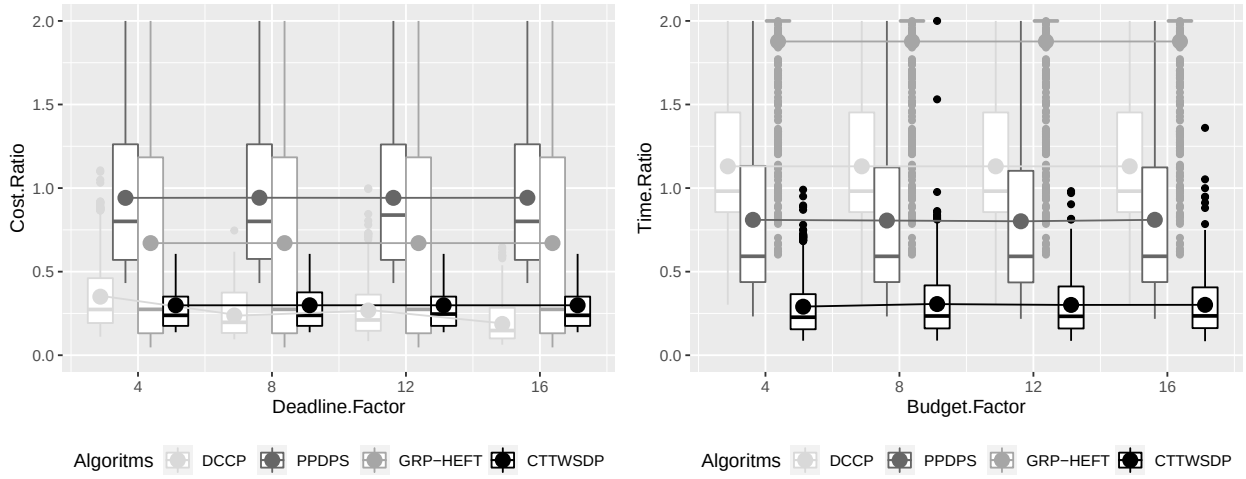
- Cost Ratio (CR): The CR of a scheduling algorithm is overall execution cost divided by the user-defined budget (B), and determined by equation (2.18) .
- Time Ratio (TR): In a similar way, the TR metric is defined as the ratio between the overall makespan and the user-defined deadline (δ), and determined by equation (2.19).
- Success Rate (SR): The SR of a scheduling algorithm is defined as the ratio between the number of ran simulations that successfully met both deadline and budget constraints (denoted by NB success), and the total number of experiments (denoted by NB_{Exp}). It is determined by equation (2.20).

3.4 Simulation Results and Analysis with ANOVA plus Tukey-Kramer post hoc test

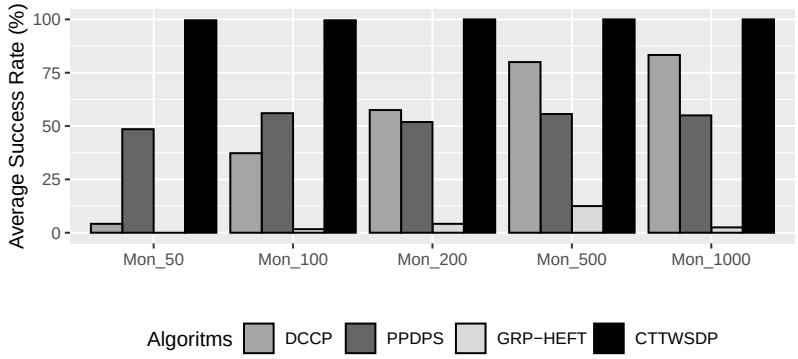
3.4.1 Performance for MONTAGE workflow

The results obtained for MONTAGE workflow are presented in Figure 3.3, for both cost and time efficiency, and in Figure 3.3c, for success rate. For time efficiency, CTTWSDP has better performance than the other algorithms; apart from a tiny number of outliers, CTTWSDP is always in the deadline (Figure 3.3b). PPDPS has more than 50% of schedules in the deadline, DCCP has a little bit less than 50% in the deadline, while GRP-HEFT is almost always out of the deadline. For cost efficiency (Figure 3.3a), DCCP and CTTWSDP have approximatively the same performance and are almost always in the budget, and they are better than the other. GRP-HEFT has a little bit less than 75% of schedules in the budget, while PPDPS has more than 50% of schedules in the budget. We realize that nor the variation of the Deadline Factor, neither the one of the Budget Factor have a significant impact on the performance of the algorithms.

In terms of average success rate for MONTAGE workflow, CTTWSDP, PPDPS, DCCP and GRP-HEFT algorithms recorded respectively 99.83%, 53.42%, 52.46% and 4.17% (Figure 3.3c).



(a) Cost Efficiency (Deadline Factor vs. Cost Ratio) (b) Time Efficiency (Budget Factor vs. Time Ratio)



(c) Success rate (%) for MONTAGE workflow.

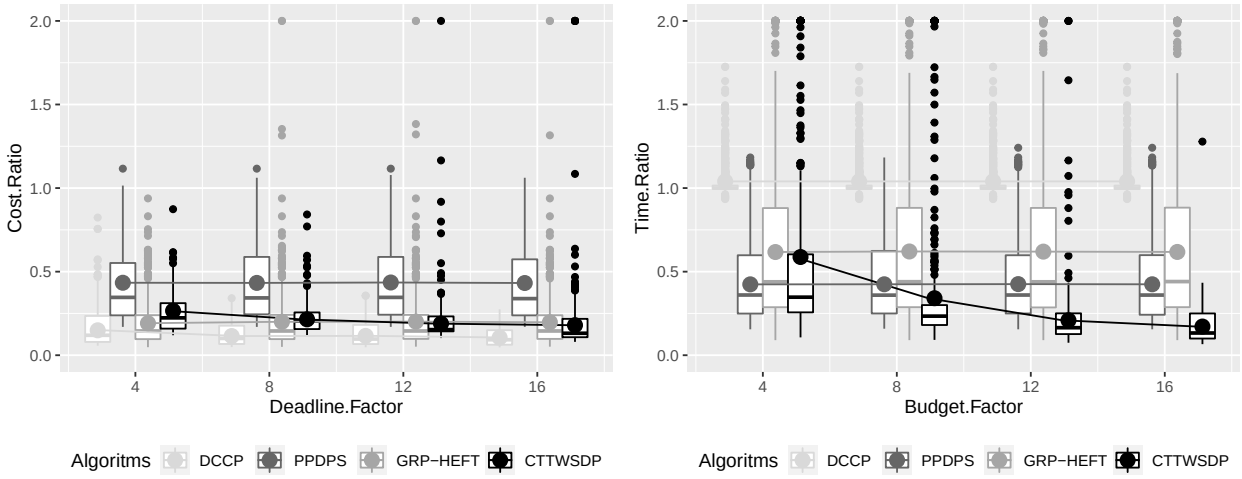
Figure 3.3: Cost efficiency, time efficiency, and success rate (%) of CTTWSDP vs DCCP, PPDPS and GRP-HEFT for MONTAGE workflow.

The low SR of GRP-HEFT, and event of DCCP, is due to the inability to be in the deadline for most of the cases. Furthermore, we observe that the workload of MONTAGE workflow has an impact over DCCP and GRP-HEFT algorithms, while CTTWSDP and PPDPS have a stable behaviour regardless of the variation of the workload.

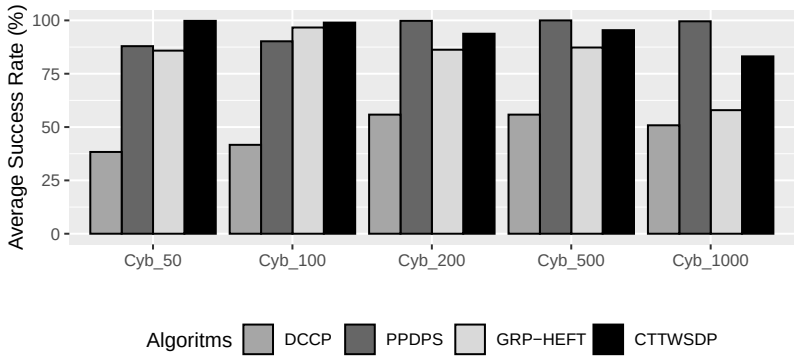
3.4.2 Performance for CYBERSHAKE workflow

The results obtained for CYBERSHAKE workflow are presented in Figure 3.4, for both cost and time efficiency, and in Figure 3.4c, for success rate. While for cost efficiency all the algorithms are almost 100% of schedules in the budget (Figure 3.4a), for time efficiency, unless DCCP algorithm which has about 75% of schedules in the deadline, the other algorithms are more than 75% always in the deadline (Figure 3.4b). We realize that for CYBERSHAKE, the variation of both the Deadline Factor and the Budget Factor have an impact on the performance of the CTTWSDP algorithm, its effectiveness grows with in the same order (Figure 3.4b).

In terms of average success rate for CYBERSHAKE workflow, CTTWSDP, PPDPS, DCCP



(a) Cost Efficiency (Deadline Factor vs. Cost Ratio) (b) Time Efficiency (Budget Factor vs. Time Ratio)



(c) Success rate (%) for CYBERSHAKE workflow.

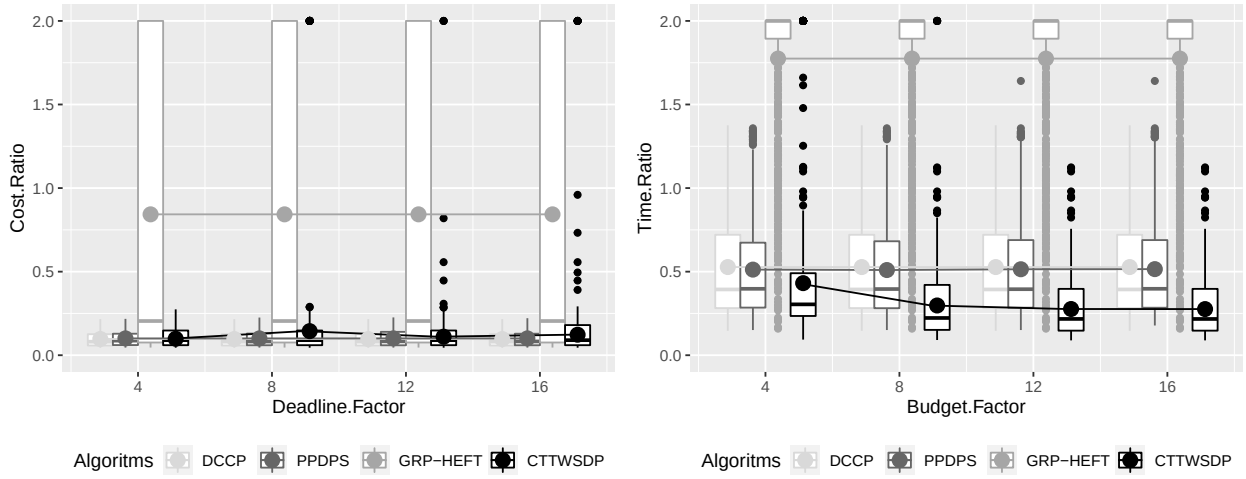
Figure 3.4: Cost efficiency, time efficiency, and success rate (%) of CTTWSDP vs DCCP, PPDPS and GRP-HEFT for CYBERSHAKE workflow.

and GRP-HEFT algorithms recorded respectively 94.21%, 95.50%, 48.50% and 82.79% (Figure 3.4c).

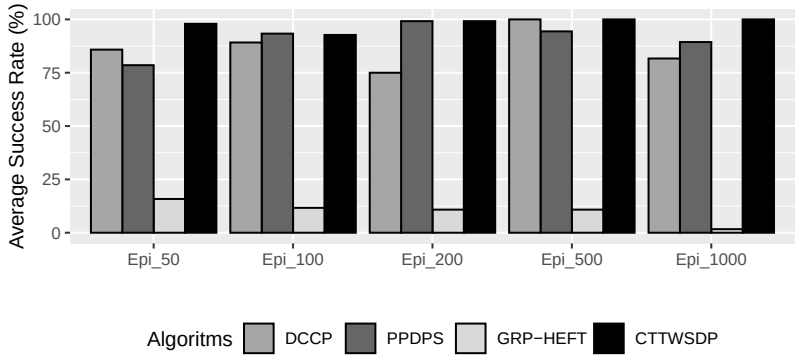
3.4.3 Performance for EPIGENOMICS workflow

The results obtained for EPIGENOMICS workflow are presented in Figure 3.5, for both cost and time efficiency, and in Figure 3.5c, for success rate. For time efficiency, unless GRP-HEFT algorithm which is out of the deadline for most of the schedules, the other algorithms are more than 75% always in the deadline (Figure 3.5b). For cost efficiency (Figure 3.5a), unless GRP-HEFT algorithm which has a little more than 50% of schedules in the budget, the other algorithms are almost 100% of schedules in the budget. As in the case of MONTAGE, we realize that nor the variation of the Deadline Factor, neither of the Budget Factor have a significant impact on the performance of the algorithms.

In terms of average success rate for EPIGENOMICS workflow, CTTWSDP, PPDPS, DCCP and GRP-HEFT algorithms recorded respectively 97.96%, 90.96%, 86.33% and 10.17% (Figure



(a) Cost Efficiency (Deadline Factor vs. Cost Ratio) (b) Time Efficiency (Budget Factor vs. Time Ratio)



(c) Success rate (%) for EPIGENOMICS workflow.

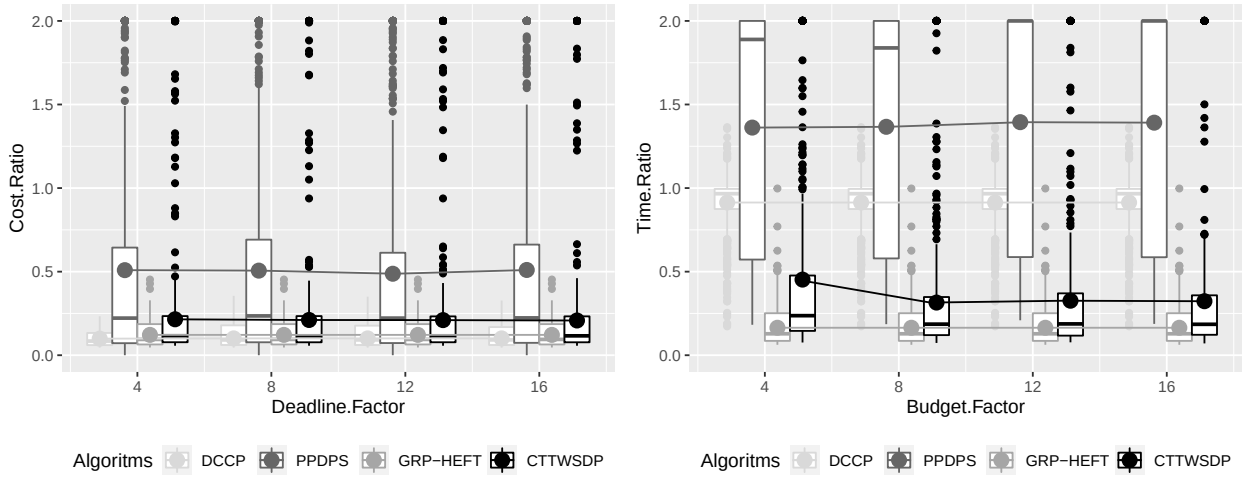
Figure 3.5: Cost efficiency, time efficiency, and success rate (%) of CTTWSDP vs DCCP, PPDPS and GRP-HEFT for EPIGENOMICS workflow.

3.5c). The low SR of GRP-HEFT is due to the inability to be in the deadline for most of the cases.

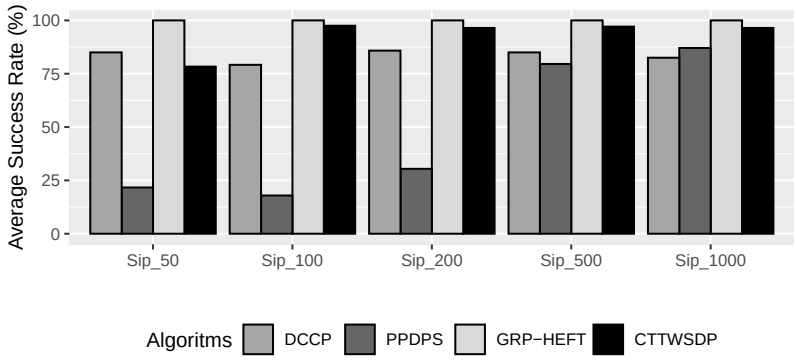
3.4.4 Performance for SIPHT workflow

The results obtained for SIPHT workflow are presented in Figure 3.6, for both cost and time efficiency, and in Figure 3.6c, for success rate. While for cost efficiency all the algorithms realize very good performance (Figure 3.4a), for time efficiency, PPDPS algorithm has less than 50% of schedules in the deadline, DCCP has about 75% of schedules in the deadline, and CTTWSDP and GRP-HEFT have towards 100% of schedules in the deadline (Figure 3.4b).

In terms of average success rate for SIPHT workflow, CTTWSDP, PPDPS, DCCP and GRP-HEFT algorithms recorded respectively 93.17%, 47.33%, 83.50% and 100% (Figure 3.6c). We inferred by looking at Figure 3.6c Furthermore, we observe that the workload of SIPHT workflow has an impact over PPDPS algorithms, while CTTWSDP, DCCP and GRP-HEFT have a stable behaviour regardless of the variation of the workload.



(a) Cost Efficiency (Deadline Factor vs. Cost Ratio) (b) Time Efficiency (Budget Factor vs. Time Ratio)



(c) Success rate (%) for SIPHT workflow.

Figure 3.6: Cost efficiency, time efficiency, and success rate (%) of CTTWSDP vs DCCP, PPDPS and GRP-HEFT for SIPHT workflow.

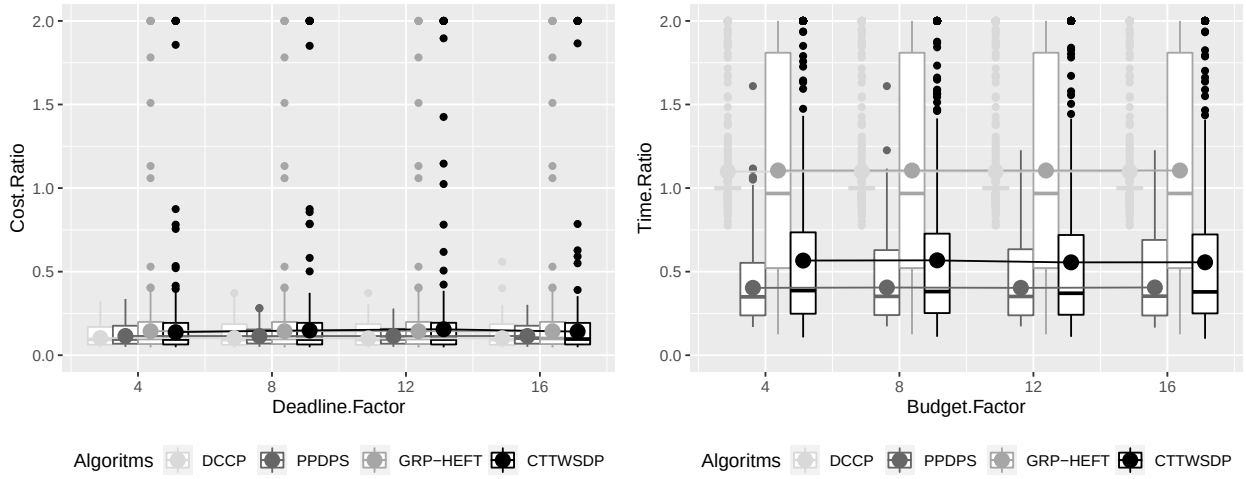
3.4.5 Performance for LIGO workflow

The results obtained for LIGO workflow are presented in Figure 3.7, for both cost and time efficiency, and in Figure 3.7c, for success rate. While for cost efficiency all the algorithms have very good performance (Figure 3.7a), in the case of time efficiency, GRP-HEFT algorithm has less than 50% of schedules in the deadline, DCCP has about 75% of schedules in the deadline, and CTTWSDP and PPDPS have almost 100% of schedules in the deadline (Figure 3.7b).

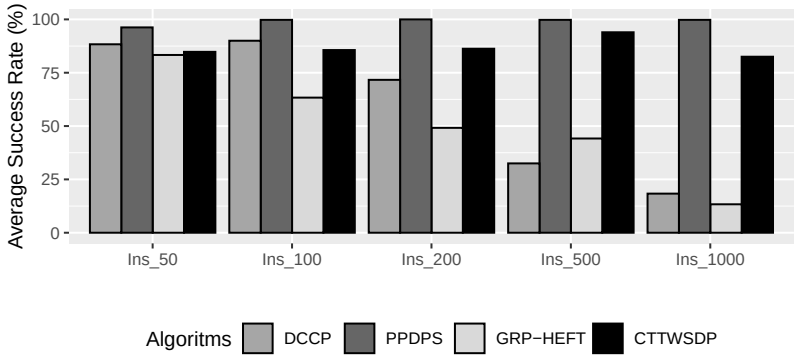
In terms of average success rate for LIGO workflow, CTTWSDP, PPDPS, DCCP and GRP-HEFT algorithms recorded respectively 86.63%, 99.12%, 60.17% and 50.00% (Figure 3.7c).

3.4.6 Performance summary

Table 3.5 presents the total SR of each algorithm. In summary, CTTWSDP has a global average of SR that is between **17.09%** − **44.80%** higher than the other algorithms as depicted in Table 3.5.



(a) Cost Efficiency (Deadline Factor vs. Cost Ratio) (b) Time Efficiency (Budget Factor vs. Time Ratio)



(c) Success rate (%) for LIGO workflow.

Figure 3.7: Cost efficiency, time efficiency, and success rate (%) of CTTWSDP vs DCCP, PPDPS and GRP-HEFT for LIGO workflow.

From the Table 3.5 we see that CTTWSDP has a overall mean of SR greater than the one of the other algorithms. However, to know if that difference is significant from each of the three other algorithms in terms of SR (i.e. customer satisfaction), we have conducted statistical analysis. Since we have more than two algorithms to compare, we have conducted ANOVA test[1] [93, 104] to see if their is a significant difference between CTTWSDP algorithm and the three other algorithms. We consider two alternatives hypotheses for comparison.

- First hypothesis (H_0): There is no difference between the proposed CTTWSDP algorithm and the three other algorithms.
- Second hypothesis (H_1) : There is a difference between the proposed CTTWSDP algorithm and the three other algorithms.

A confidence interval (CI) of 95% has been used for the test. Table 3.6 presents the result of the ANOVA test. As we can see in Table 3.6b, we have F statistical > F critical (and moreover

[1]https://www.excel-easy.com/examples/anova.html

Table 3.5: Success rate summary for the five scientific workflows realized by the four dynamics algorithms. The total average of SR for CTTWSDP is between 17.09% – 44.80% higher than other algorithms.

Workflow	CTTWSDP	PPDPS	DCCP	GRP-HEFT
MONTAGE 50	**99.58**%	48.54%	4.17%	0.00%
MONTAGE 100	**99.58**%	56.04%	37.29%	1.67%
MONTAGE 200	**100**%	51.88%	57.50%	4.17%
MONTAGE 500	**100**%	55.63%	80.00%	12.50%
MONTAGE 1000	**100**%	55.00%	83.33%	2.50%
CYBERSHAKE 50	**99.79**%	87.92%	38.33%	85.83%
CYBERSHAKE 100	**98.96**%	90.21%	41.67%	96.67%
CYBERSHAKE 200	93.75%	**99.79**%	55.83%	86.25%
CYBERSHAKE 500	95.42%	**100**%	55.83%	87.29%
CYBERSHAKE 1000	83.13%	**99.58**%	50.83%	57.92%
EPIGENOMICS 50	**97.92**%	78.54%	85.83%	15.83%
EPIGENOMICS 100	92.71%	**93.33**%	89.17%	11.67%
EPIGENOMICS 200	**99.17**%	**99.17**%	75.00%	10.83%
EPIGENOMICS 500	**100**%	94.38%	**100**%	10.83%
EPIGENOMICS 1000	**100**%	89.38%	81.67%	1.67%
SIPHT 50	78.33%	21.67%	85.00%	**100.00**%
SIPHT 100	97.50%	17.92%	79.17%	**100.00**%
SIPHT 200	96.46%	30.42%	85.83%	**100.00**%
SIPHT 500	97.08%	79.58%	85.00%	**100.00**%
SIPHT 1000	96.46%	87.08%	82.50%	**100.00**%
LIGO 50	84.79%	**96.25**%	88.33%	83.33%
LIGO 100	85.63%	**99.79**%	90.00%	63.33%
LIGO 200	86.25%	**100**%	71.67%	49.17%
LIGO 500	93.96%	**99.79**%	32.50%	44.17%
LIGO 1000	82.50%	**99.79**%	18.33%	13.33%
Mean	**94.36**%	77.27%	66.19%	49.56%
Standard deviation	**2.42**	21.98	15.77	38.11

p-value < 0.05). Thus, we can reject the null hypothesis and conclude that we have sufficient evidence to say that at least one of the SR of an algorithm is different from the others.

To determine exactly which algorithm is significantly better than the others in terms SR, we have performed a Tukey-Kramer test [105]. From the ANOVA test and using the Critical Values of the Studentized Range[2] (0.05 level), we have determined [105] the Q value and the Q critical value (respectively 3.6976 and 20.50) for the Tukey-Kramer test. Table 3.7 presents the results of the Tukey-Kramer test between the SR achieve by the four algorithms CTTWSDP, PPDPS, DCCP, and GRP-HEFT.

From the result of Table 3.7, CTTWSDP significantly outperform GRP-HEFT and DCCP algorithms, while the test reveals that there is no significant difference with PPDPS.

[2]http://davidmlane.com/hyperstat/sr_table.html

Table 3.6: ANOVA test result comparing the SR of the proposed CTTWSDP algorithm with the three other algorithms (PPDPS, DCCP, and GRP-HEFT)

Group	***Count***	***Sum***	***Average***	***Variance***
CTTWSDP	25	2358.97	**94.36**	**45.32**
PPDPS	25	1931.68	77.27	715.61
DCCP	25	1654.78	66.19	645.18
GRP-HEFT	25	1238.96	49.56	1668.00

(a) Summary of input

Source of Variation	***SS***	***df***	***MS***	***F statistical***	***P-value***	***F critical***
Between Groups	26623.23	3	8874.41	11.55	1.57E-06	2.70
Within Groups	73778.91	96	768.53			
Total	100402.15	99				

(b) ANOVA test result

Table 3.7: Tukey-Kramer test for Pairwise of algorithms comparing the SR of the proposed CTTWSDP algorithm with the three other algorithms (PPDPS, DCCP, and GRP-HEFT)

Parameter	***Value***
$Q_{.05}$	3.6976
Q critical value	**20.50**

(a) Parameters of the test

Comparison	***Mean Diff***	***Abs. Mean Diff***	***Is Significant?*** *(Abs. Mean Diff ≥ Q critical value?)*
CTTWSDP vs PPDPS	17.0916	17.0916	NO
CTTWSDP vs DCCP	28.1676	**28.1676**	**YES**
CTTWSDP vs GRP-HEFT	44.8004	**44.8004**	**YES**
PPDPS vs DCCP	11.076	11.076	NO
PPDPS vs GRP-HEFT	27.7088	27.7088	**YES**
DCCP vs GRP-HEFT	16.6328	16.6328	NO

(b) Tukey-Kramer test for Pairwise of algorithms

Furthermore, the following remarks have to be mentioned:

- The performance of CTTWSDP contrasts greatly with both GRP-HEFT and DCCP because of their high rate of failure due to deadline violation which can be inferred by looking at Figures 3.3b, 3.5b, 3.4b and 3.7b. However, DCCP performs better than GRP-HEFT.
- The structure of the workflow has a significant impact on the performance of GRP-HEFT, PPDPS, and DCCP algorithms. On the other hand, the CTTWSDP algorithm has good behavior regardless of the structure of the workflow as well as the workload.
- Overall, the CTTWSDP algorithm is more effective than the other algorithms no matter the type of workflow and the workload, as proved by the above statistical tests.
- The workload of SIPHT and LIGO workflows have a significant impact on the performance of PPDPS and GRP-HEFT algorithms respectively.

- The relatively low success rate of the GRP-HEFT and DCCP algorithms are mostly due to the violation of the user-defined deadline.

3.5 Quality improvement analysis of the CTTWSDP algorithm

In this section we are analysing the quality improvement incurred by the IRITR evaluation. Since the IRITR aims at determining a suitable range of instance types, we are varying the set of instance type available for the experiment.

We are considering four different sub-sets of instance types of those presented in Table 2.2:

1. From *m3.medium* to *m5.8xlarge* ($M3\_M - M5\_8XL$): the six slowest instances;
2. From *m4.4xlarge* to *m5.24xlarge* ($M4\_4XL - M5\_24XL$): the six fastest instances;
3. From *m4.xlarge* to *m5.12xlarge* ($M4\_XL - M5\_12XL$): the six instances of the middle;
4. From *m3.medium* to *m5.24xlarge* ($M3\_M - M5\_24XL$): all the ten instances.

We fixed $\alpha = 16$ and $\beta = 16$, and still conducted 100 experiments for each workflow structure (20 for each of the five sizes). The purpose of these experiments is to determine how well our algorithm is able to evaluate the instance types range in a large list, for good scheduling, since cloud providers often offer a large list of different types of instances.

In order to highlight the level of relevance of the conceptual strength of the IRITR evaluation, we conducted a sensitivity analysis. To this end, we removed the IRITR evaluation from the CTTWSDP algorithm, which corresponds to ignoring lines 9 and 10 of algorithm 3 and line 2 of algorithm 4. We conducted the experiments both on the resulting algorithm, named CTTWSDP without IRITR and denoted by CTTWSDP-W-IRITR, and on CTTWSDP.

The results of the experiments summarized in Table 3.8 show clearly the impact of the IRITR in the scheduling when the set of instances type is large, or simply when there are too slower (cheaper) instances or too fast (expansive) instances, which correspond to the first and the two last sets of instances. CTTWSDP outperforms CTTWSDP-W-IRITR in terms of SR average of 4.93% to 28.80%. CTTWSDP and CTTWSDP-W-IRITR score respectively 96.47% and 77.50% of total SR average.

For CTTWS algorithm, there were cases in which the algorithm version without IRITR evaluation (CTTWS-W-IRITR) realized better SR than CTTWS. Also, the difference of SR average was at most of 10.2% between CTTWS and CTTWS-W-IRITR. That is not the case in this improved version. This clearly highlights the conceptual strength of the IRITR evaluation presented in this chapter.

Table 3.8: Success rate summary for the five scientific workflows and the different set of instance types. CTTWSDP can have a total average of SR that is higher than the one of CTTWSDP-W-IRITR from 4.93% to 28.80% according to the available set of instance types.

	M3_M - M5_8XL		***M4_4XL - M5_24XL***		***M4_XL - M5_12XL***		***M3_M - M5_24XL***	
Workflow	CTTWSDP	CTTWSDP-W-IRITR	CTTWSDP	CTTWSDP-W-IRITR	CTTWSDP	CTTWSDP-W-IRITR	CTTWSDP	CTTWSDP-W-IRITR
MONTAGE	86.67%	33.33%	100%	99.33%	100%	40.00%	100%	40.00%
CYBERSHAKE	100%	51.33%	100%	97.33%	100%	80.67%	98.67%	40.00%
EPIGENOMICS	100%	99.33%	100%	96.00%	100%	99.33%	100%	97.33%
SIPHT	96.00%	79.33%	95.33%	79.33%	95.33%	82.67%	95.33%	74.67%
LIGO	83.33%	83.33%	92.67%	91.33%	91.33%	92.67%	94.67%	92.67%
Mean	93.20%	69.33%	97.60%	92.67%	97.33%	79.07%	97.73%	68.93%

Moreover, the limitation of the number of VMs to lease has clearly improved the performance of the algorithms. In fact, the performance of the CTTWS algorithm for SIPHT workflow was very poor, almost 0% in all the cases. That is not the case in this version (see Figures 3.6 and Table 3.5).

Conclusion

In this chapter, we have proposed a cost/time effective scheduling algorithm in the cloud, named Cost-Time Trade-off efficient Workflow Scheduling with Dynamic provisioning (CTTWSDP), which is an improved version of the CTTWS algorithm presented in the chapter 2. The CTTWSDP algorithm strives to minimize both execution cost makespan under user-defined budget and deadline constraint, with a stable behaviour regardless of the structure and the workload of the workflow. The main improvement are the introduction of a dynamic VMs provisioning strategy with the limitation of the number of VMs to lease, and the enhancement of the Implicit Requested Instance Types Range (IRITR) evaluation by introducing the average width of the workflow structure in the formulas.

The limitation of the number of VMs and the enhancement of the IRITR greatly contributed to the effectiveness of CTTWSDP. For instance, in our previous chapter, the performance of the CTTWS algorithm for SIPHT workflow is very poor. Almost 0% of SR in all the cases due to the big number of tasks at the root of SIPHT, which is not the case for CTTWSDP.

The results of the experiments show that CTTWSDP produces a global average of success rate that is between 17.09% and 44.80% higher than state-of-the-art algorithms. In addition, the standard deviation of success rate across the different types and workloads of workflow obtained by CTTWSDP is smaller than the ones obtained by the other algorithms from 13.35 to 35.69; proving that CTTWSDP is more stable in its performance no matter the type and the workload of workflow. Furthermore, ANOVA test along with pairwise tests using Tukey-Kramer have been conducted. At overall, the result of the statistical tests reveals that CTTWSDP is significantly more effective than two of the baseline algorithms, namely GRP-HEFT and DCCP, while for PPDPS the out-performance of CTTWSDP is not statistically significant. The selection of a sub-set of instances via the IRITR technique helped producing better results. Furthermore, suitable number of VMs determined by the average width of the workflow helped producing better results. Both techniques have been designed by exploiting the structural properties of workflow and user requirements. Therefore, our hypotheses have been clearly validated by the contributions presented in this chapter.

In the next chapter, in addition to the minimization of the execution cost and time, we deal with the minimization of energy consumption.

CHAPTER 4

Energy-efficient workflow scheduling strategies based on workflow structures under Budget and Deadline constraints

One new trend in Information technology is the usage of Cloud computing environments to perform scientific workflow applications. Workflow scheduling is the main issue in workflow management and known as an NP-complete problem. Scientific workflows are generally complex and have different characteristics. The structure of a workflow can have a significant impact on the result of a scheduling algorithm. Therefore, a scheduling algorithm should not ignore or consider only one particular workflow structure but adapt to different possible structures. Apart from the user-defined deadline and budget, energy consumption is a major concern in the cloud. In this chapter, we design and evaluate three workflow scheduling algorithms that take advantage of the structural properties of the workflow. A new scheduling algorithm, namely, Structure-based Multi-objective Workflow Scheduling with an Optimal instance type (SMWSO) is presented, which introduces new concepts. The optimal number of VMs evaluation along with the optimal instance type evaluation, helping to deal with the issue of resource heterogeneity, and to avoid resource wastage by limiting the number of VMs to provision while giving relatively good performances. A version with heterogeneous instance types (denoted SMWSH) is proposed in order to highlight the strength of the just mentioned concepts. The third algorithm called Structure-based Cost-Time trade-off and Energy efficient Workflow Scheduling (SCTTEWS) algorithm is an incrementation of our CTTWSDP algorithm, which handles the minimization of energy consumption in addition to the minimization of the execution cost and time. Comparative experimentation have been done through simulations against the state-of-the-art algorithm REEWS. The analysis of the results supported by ANOVA along with pairwise tests using Tukey-Kramer proves the out-performance of our proposals in terms of energy-saving compared to the REEWS algorithm in 80% of workflow/workload scenarios. Among others, SMWSO, SMWSH and SCTTEWS scored almost equally the highest energy-saving for the different workflow and workload. However, SMWSO save more than 50% overall energy compared to other algorithms, followed by SMWSH, and then SCTTEWS. As for the success rate, even though SMWSO scored at overall the highest success rate, statistical tests proved that there is no significant difference between the four algorithms in terms of user satisfaction. The results revealed the ability of SMWSO to deal effectively with the heterogeneous nature of cloud environments, and the com-

plex structures of workflows.

Introduction

Scientific workflows have very complex structures [5, 4] that can significantly impact the outcome of a scheduling strategies. Some of the important parts of a workflow structure have been widely investigated, for instance the Critical Path (CP) (with the slack time reclamation/DVFS strategy), the distribution[1] tasks (with task duplication strategy), and the sequential and parallel tasks (with the tasks merging and slack time reclamation/DVFS strategies).

However, little or almost no work combines all these strategies to see what it can achieve. In addition, the width of the workflow (the distribution of the number of tasks by level of workflow from entry to exit) is almost not studied, while important because of the precedence constraints existing in the workflow. In fact, it is unlikely to use more VMs than the largest width of the workflow for its execution. It is then important to know or determine the optimal number and the types of VMs to use during the execution of a workflow. To the best of our knowledge, there is no work focused on determining suitable instance types set or a number of VMs instances in advance with an analytical approach. Some solutions use a *naive determination approach* in which it is at the end that one realize which types and the number of VMs have been used, leading in more cases to a wastage (too many provisioned VMs that are less utilized). Others are time consuming determination approaches, like greedy determination [44] and paths-based clustering determination [21][45]. The paths-based clustering approach is better than the greedy one, however, its complexity and effectiveness are compromised if the workflow graph is strongly connected. Moreover, most of the solutions in the literature are effective only for a few types of workflow, while the types and structures of workflow are very complex and varied [4]. This is not conform with the recommendation [4][41] of designing scheduling strategies that are effective no matter the type of workflow.

We advocate that homogeneity can produce better result if the good instance is chosen for the workflow execution, as it has been some how proved in the two last chapters. We further advocate that a suitable number of VMs if determined, can help not only to produce better results, but also upgrade the VM utilization Maximization and the Energy Consumption Minimization as well as the Workload Maximization. To respond to our just mentioned propositions, we designe and present in this chapter three scheduling algorithms, Structure-based Cost-Time trade-off and Energy efficient Workflow Scheduling (SCTTEWS), Structure-based Multi-objective Workflow Scheduling with an Optimal instance type (SMWSO), and Structure-

[1] tasks having more than one child, see Figure 4.1

based Multi-objective Workflow Scheduling with Heterogeneous instance types (SMWSH).

In the following sections, after the recall of the problem and some investigation on the influence of the structures of the workflows, we present our three heuristics, and finally we present the performances analysis of our three proposals against Reliability and Energy Efficient Workflow scheduling (REEWS), a recent state-of-the-art algorithm [21]. The choice of REEWS is because it uses a clustering technique to determine the number of VMs to be used and the DVFS to minimize energy consumption.

4.1 Modelling of the workflow scheduling problem

In this section we present the cloud resource model, the power and energy models, the workflow model, and the problem formulation.

4.1.1 Cloud computing model

The cloud computing model presented in chapter 3, section 3.1.1, is the same used in this chapter. Since energy consumption is considered in this chapter, the next sub-section presents the power and energy evaluation.

4.1.2 Power and Energy models

In terms of energy consumption among system components, processors consume typically the largest portion [20, 106]. Hence, we will focus on energy consumption of processors. A processor consumes energy either idle or while running a task. The power consumed by a processor p_k during its runtime, noted P^k, is expressed by equation (4.1) [20, 107].

$$P^k(u_k(t)) = P^k_{idle} + (P^k_{max} - P^k_{idle}) \times u_k(t), \tag{4.1}$$

where P^k_{idle} and P^k_{max} are the power consumed by the processor when idle and at 100% utilization respectively, whereas $u_k(t)$ is the utilization rate of the processor, which is a function of the time. Therefore, the total energy consumption of a processor p_k over a period of time [t0, t1] can be defined as an integral of the power consumption function over the same period as expressed in equation (4.2).

$$E^k = \int_{t1}^{t0} P^k(u_k(t))dt \tag{4.2}$$

Then the overall energy consumption (E_{total}) on all the P VMs is simply the sum of all the energy consumption.

4.1.3 Workflow model

Workflow as a Directed Acyclic Graph (DAG)

The workflow model presented in chapter 2, section 2.1.2, is the same considered in this chapter. Scientific workflows are still represented as DAGs.

Definitions

In addition to the definitions related to the workflow provided in section 2.1.2, we give the following.

The communication to computation ratio (CCR) of a workflow is the ratio of its average communication cost to its average computational cost on the targeted system [43] and is given by:

$$CCR_G = \frac{\sum_{t_i \in WT} \overline{TT(i,j)}}{\sum_{t_i \in WT} \overline{ET(i,k)}}, t_j \in Succ(t_i); 1 <= k <= K \tag{4.3}$$

The CCR can be used as an indicator of the type of the workflow among the following: communication-intensive (data-intensive) when CCR is greater, or computationally intensive (CPU intensive) when CCR is smaller.

Zhang and Chakrabarty [55] define the reliability of the system as the probability of executing workflow tasks without any failure. Two type of faults can occur during the execution of an application due to crashing of hardware, flaws in software, high temperature attained by the machine, etc. We have permanent and transient faults. The probability of occurrence of transient faults is much more than that of permanent faults [55]. Zhang and Chakrabarty [55] model transient faults following Poisson distribution as follow:

$$\begin{aligned} \lambda(f_{r,op}) &= \lambda_0 \times F(f_{r,op}) \\ &= \lambda_0 \times 10^{\frac{d * (1 - f_{r,op})}{f_{r,min}}} \end{aligned} \tag{4.4}$$

where $f_{r,op}$ denotes the operating frequency, $f_{r,min}$ the lowest frequency, λ_0 the initial fault rate at maximum voltage/frequency, $F(f_{r,op})$ a strictly decreasing function of frequency and

d (> 0) a constant. Fault rate is maximum (minimum reliability) at lowest frequency $f_{r,min}$ (which is most suitable frequency for energy conservation).

The reliability of the system which is the probability of execution of task t_i without any failure is determined as follow:

$$Rel_{t_i}(f_{r,op}) = e^{-\lambda(f_{r,op})*\frac{ET(i,k)}{f_{r,op}}} \tag{4.5}$$

For the n tasks of the workflow we have:

$$Rel_G = \prod_{i=1}^{n} Rel_{t_i}(f_{r,op}) \tag{4.6}$$

4.1.4 Problem Formulation

The role of a workflow scheduler is to determine an execution order of the workflow tasks, and the VM onto which to assign each task. That mapping of tasks onto VMs have to satisfy some requirements of the user and the cloud provider.

In this chapter, the targeted objectives of the workflow scheduler are the reduction of the overall execution cost and execution time, as well as the energy consumption of the system. The constraint remain the user-defined deadline δ and the user-defined budget B.

The question to deal with is: *how to build a workflow scheduling algorithm, able to dynamically provision VMs for tasks execution in order to reduce the overall execution cost and execution time as well as the energy consumption of the system, in the respect of the user-defined budget and deadline?*

The problem can be formulated as a mathematical optimization problem:

$$\begin{cases} Minimize(E_{total}) \\ Reduce(M_G) \\ Reduce(Cost_G) \\ Subject\ to\ M_G \leq \delta\ and\ Cost_G \leq B \end{cases} \tag{4.7}$$

4.2 Workflow structure influence

Intuitively, in the last chapter, we have employed the average width of the workflow as a suitable number of VMs to use for the execution of the workflow. However, due to the big complexity

and diversity of the structures of workflows, we conducted some experiments to investigate the influence of workflows structure over the scheduling performance.

The Figure 4.1 presents the basic structures of a workflow [5].

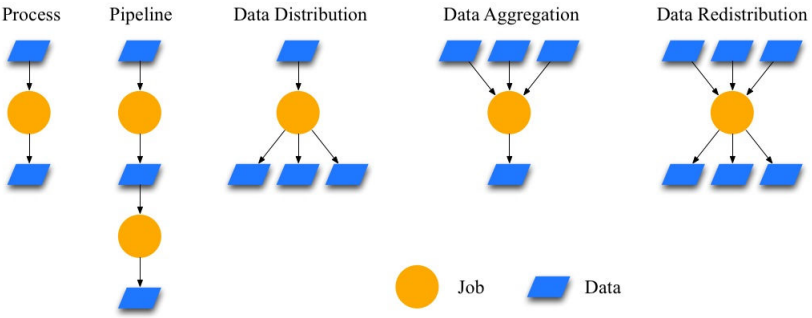


Figure 4.1: Basic workflow structures [5].

4.2.1 Workflow width

Analysis of the optimal number of VMs for the execution of a workflow: Figure 4.2 shows the structure of a Montage workflow with twenty tasks and their dependencies. In this figure, the left column shows level numbers calculated by equation (4.8), while the right column is the number of tasks in each level that we call the level width ($levelWidth(l)$). In this example the largest width is 6 which corresponds to the level 2 ($levelWidth(2) = 6$).

$$LN(t_j) = 1 + \max_{t_i \in pred(t_j)} \{LN(t_i)\}, \tag{4.8}$$

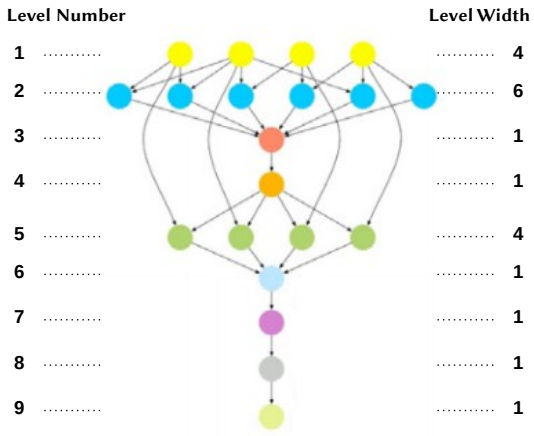
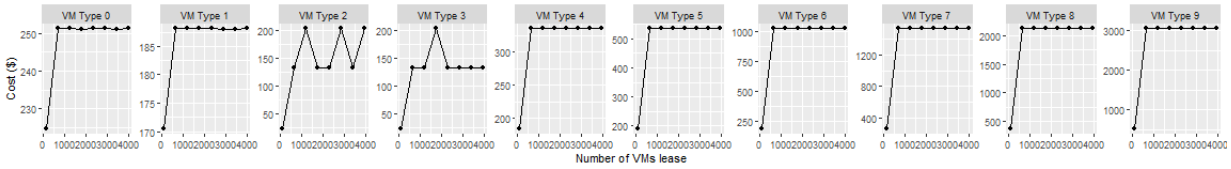


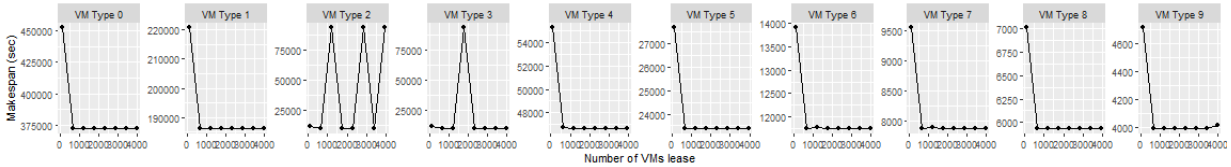
Figure 4.2: Workflow width distribution.

Obviously, it is unlikely to use more VMs than the largest width of the workflow for its execution. But at the same time how many VMs is suitable for the execution of the workflow?

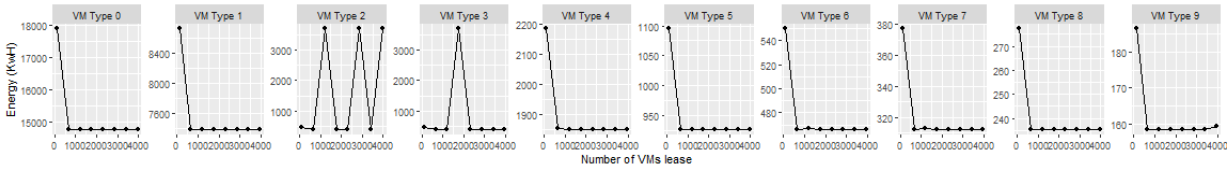
To analyse that we conducted some experiments using HEFT algorithm [82], in which we have varied the number of VMs supplied and the instance type among the ones of Table 2.2. The experiment setup was the same as the one described in section 2.3. In each experiment, all the VMs was of a same instance type taken among the ten instance types of the Table 2.2. The experiments was conducted using Montage, CyberShake, Epigenomics, Sipht and LIGO (Inspiral) workflows. For each of the workflows, five different workloads with respectively 50, 100, 200, 500 and 1000 tasks were considered. Since the analysis results were the same for all the workflows and for the five different workloads, we just presented here the workload of 1000 tasks per workflow (see Figure 4.3 in this section, and Figures A.1, A.2, A.3, A.4 in the section A.1).



(a) Cost per number of VMs leased



(b) Makespan per number of VMs leased



(c) Energy per number of VMs leased

Figure 4.3: Cost, Makespan and Energy per number of VMs leased for Montage_1000.

By analysing the results of the experiment we found a relation that can provide an optimal number of VMs (ONVM) regardless the instance type of VM, that produces a good compromise among the cost, the makespan and the energy consumption. Assuming the distribution of width of the different levels of the workflow, that number is given by the equation (4.9).

$$ONVM(wf) = \begin{cases} AvgW_{Wf} & , \; if \; AvgW_{Wf} \leq StdDW_{Wf}, \\ \min\{AvgW_{Wf} + StdDW_{Wf}, MaxW_{Wf}\} & , \; otherwise \end{cases} \tag{4.9}$$

Where $MaxW_{Wf}$ is the maximum, $AvgW_{Wf}$ the average and $StdDW_{Wf}$ the standard deviation of the levels' width of the workflow.

4.3 Structure based techniques

In this section we propose some optimisation techniques based on the structural properties of the workflows, that are used in our three multi-objective heuristics. Among the proposed scheduling techniques, the common ones used in these three algorithms are:

- The Entry Task Duplication Policy;
- The Pipeline Merging and Slacking;
- The optimal number of VMs determination which is used for th limitation of VMs leasing.

The specific technique used within the SMWSO algorithm is the determination of the Optimal instance type employed for resources homogeneity. In the case of the SCTTEWS algorithm, the specific techniques are:

- The (improved) Implicit Requested Instance Types Range (IRITR) evaluation;
- The weighted trade-off factors used in the trade-off function.

Each of these techniques is used in one of the main phases of a workflow scheduling algorithm that are [3]:

1. *Resources provisioning*: it consists of selecting and provisioning the compute resources that will be used to run the workflow tasks.
2. *Scheduling* or *task allocation*: it consists of mapping each task onto the best-suited resource. Therefore, it can be divided into two stages:
   (a) *Task selection*: it consists of selecting a task among the non yet scheduled tasks of the workflow. It rely on a tasks prioritization.
   (b) *task to VM mapping*: it consists of mapping the selected task onto the best-suited resource.

4.3.1 Choice of the number of VMs

Choice of the number of VMs to use

The optimal number of VMs ($optNbVMs$) to use for the execution of the workflow by the equation 4.9, and must not be exceeded during the scheduling process. That formula considers

the width distribution of the different levels of the workflow. During our investigation, we noticed that some algorithms (like HEFT [82] for instance) employ more VMs than the possible path of the workflow for the execution. That led us to the research of an optimal number of VMs to used which is dynamically determined according to the structure of the workflow.

It seems unsuitable to use more VMs than the largest width of the workflow for its execution. But at the same time, it is good to know the suitable number of VMs to use for the execution of the workflow?

For the example of Figure 4.2, the determination of the optimal number of VMs is presented in Table 4.1.

Table 4.1: Example of determination of optimal number of VMs based on Figure 4.2.

Width distribution	**Max**	**Avg**	**Std Dev.**	**optimal Nb of VMs**
4; 6; 1; 1; 4; 1; 1; 1; 1	6	2.22	1.81	$2.22 + 1.81 \simeq 4$

4.3.2 Task priority

The order of execution of workflow tasks is very important in a scheduling strategy. Any ordering strategy used for workflow tasks most take into account the precedence constraints between the tasks. One of the most used ordering strategy is the up-rang of Topcuoglu et *al.* [82], which has been improved by Wang et *al.* [108]. In our case, the tasks are ordered in a list, called ready list. Our task priority strategy is an improvement of the modified $rank_u$ proposed by Wang et *al.* [108] and is given by equation (4.10).

$$rank_u(t_i) = \begin{cases} \sigma_{exit} & , if\ t_i = t_{exit} \\ \sigma_i + outd(t_i) + \overline{OCCW(t_i)} + \max\limits_{t_j \in Succ(t_i)} \{rank_u(t_j)\} & , otherwise \end{cases} \tag{4.10}$$

where σ_i is the standard deviation of the computation time of the task t_i on the available pool of processors. The task with the highest $rank_u$ is more prioritised.

We are using just the standard deviation instead of multiplying it with the average computation time as it is done in [108]. Furthermore, we are using the average communication cost weight ($\overline{OCCW(t_i)}$) instead of the OCCW, and in addition we are using the out-degree of the task which will grant more priority to tasks having more children.

4.3.3 Entry Task Duplication Policy

The aim of the Task Duplication is to reduce the execution time of the workflow by eliminating data transfer time from the entry task [63, 108], and therefore, reduce the makespan and eventually the execution cost of the workflow. It more suitable for data intensive workflow. Task duplication can have an impact over the makespan, the cost and the energy consumption. To the best of our knowledge, the entry task duplication selection policies found in the literature does not take into account the case where there are several root tasks in the structure of the workflow. While it help reducing the makespan, it may raise a significant increase of the execution cost and even the energy consumption if the entry task is CPU intensive. It is then necessary to limit the number of duplication and use an accurate duplication policy. In our case, more than one root tasks can be duplicated according to the following policy (see Figure 4.4):

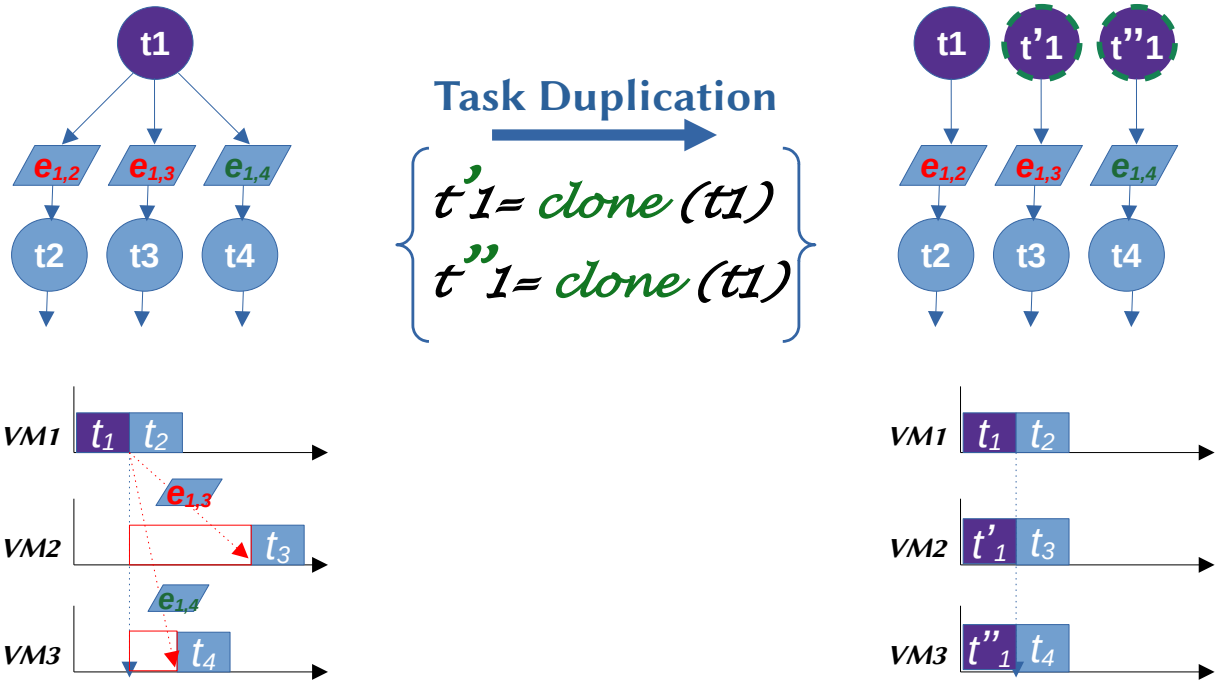


Figure 4.4: Entry Task Duplication

1. There may be task duplication only if $totalRepDue = min(optNbVMs, levelWidth(2)) - levelWidth(1) > 0$;

2. Under the condition (1.), an entry task t_i can be duplicated if it is the only parent of more than one child ($nbSingleParentChildren_i > 1$). That is, no duplication for children having several parents;

3. Assuming the $nbSingleParentChildren_i$ children ordered according to their priority, proceed to a task duplication of the $nbSingleParentChildren_i - 1$ first children (t_j) as long

it is possible:

(a) Duplicate t_i to the VM vm_p ($vm_p \in VMS, vm_p <> map(i)$) with the lowest execution cost and which fulfills the condition of equation (4.11)

$$ET(i, vm_p) < ET(i, map(i)) + TT(i, j) \tag{4.11}$$

(b) If a such VM vm_p exists, map t_j to vm_p ($map(j) = vm_p$)

(c) If not, provision a new VM $vm_{p'}$ of type $optInstType$, map t_j to $vm_{p'}$ ($map(j) = vm_{p'}$)

4. Assign the first unmapped child (among $nbSingleParentChildren_i$ children) to $map(i)$;

We will investigate the duplication on behalf of children having several parents in our future work.

4.3.4 Pipeline Merging and Slacking

The pipelines Merging and Slacking is a scheduling technique that aim at maximizing resources utilization, reducing energy consumption, and reducing execution cost eventually, through a smart management of sequential and parallel tasks.

A pipeline is a succession of tasks having exactly one parent and one child (see Figure 4.1). We consider as parallel tasks, unlike the literature [109, 97], a set of pipelines or process tasks coming from the same parent (which is a distribution task) and leading to the same child (which is an aggregation task).

Our Pipeline Merging and Slacking process is in two phases (see Figure 4.5). The first phase is to identify parallel pipelines and merge some pipelines in the group, and the second is to apply slack time reclamation to the tasks of some pipelines in each group.

Parallel pipelines grouping and merging: If the current task t_i is a distribution task (ei. $outd(t_i) > 1$):

1. Determine whether there are pipelines beginning from one of its children;

2. Construct groups of parallel pipelines;

3. In each group of parallel pipelines:

   (a) Determine the longest pipeline (the one having the highest sum of computation length ($pipeGpL_{max}$));

(b) Constitute sub-groups of pipelines in which the sum of computation length is less or equal to the length of the longest pipeline ($pipeGpSumL \leq pipeGpL_{max}$);

The pipelines in the same sub-group could then be mapped to the same VM instance without delaying the execution time, rather, it is possible to have slack times to reclaim.

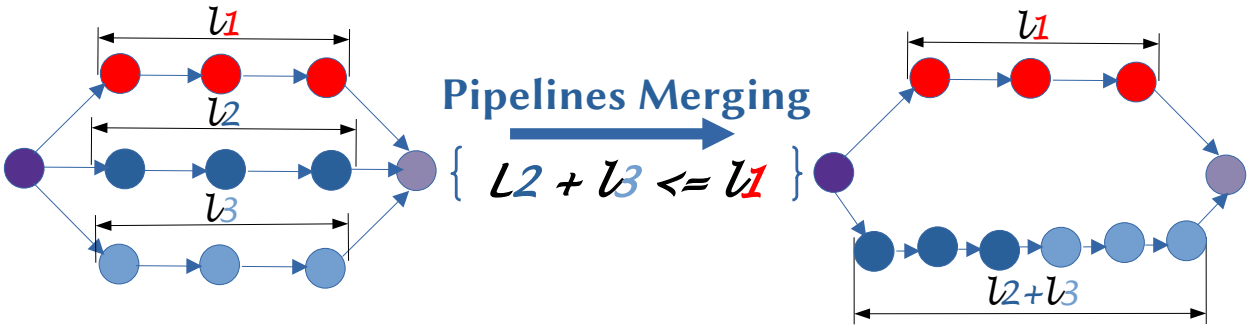


Figure 4.5: Pipelines Merging

Pipeline Slacking: The process of the Pipeline Slacking is as follow (see Figure 4.6): if the current task t_i is at the head of a pipeline :

1. Determine the slack time: $slackTime = pipeGpL_{max} - pipeGpSumL$;
2. Determine the CPU utilization rate: $cpuUtilization = pipeGpL_{max}/(pipeGpL_{max}+slackTime)$;
3. Set the CPU utilization rate of t_i to $cpuUtilization$ using the DVFS technique;
4. For all the other tasks of the current pipeline (subsequent children of t_i), and all the tasks of the pipelines in the same sub-group:
   (a) Map the task to $map(i)$;
   (b) Set the CPU utilization rate of the task to $cpuUtilization$ using the DVFS technique;

The Pipeline Merging and Slacking technique helps in the reduction of the makespan, the energy consumption, and also the execution cost.

4.4 Structure-based Cost-Time trade-off and Energy efficient Workflow Scheduling (SCTTEWS)

Here we propose an algorithm that extends the CTTWSDP algorithm by handling the minimization of energy consumption in addition to the minimization of the execution cost and time.

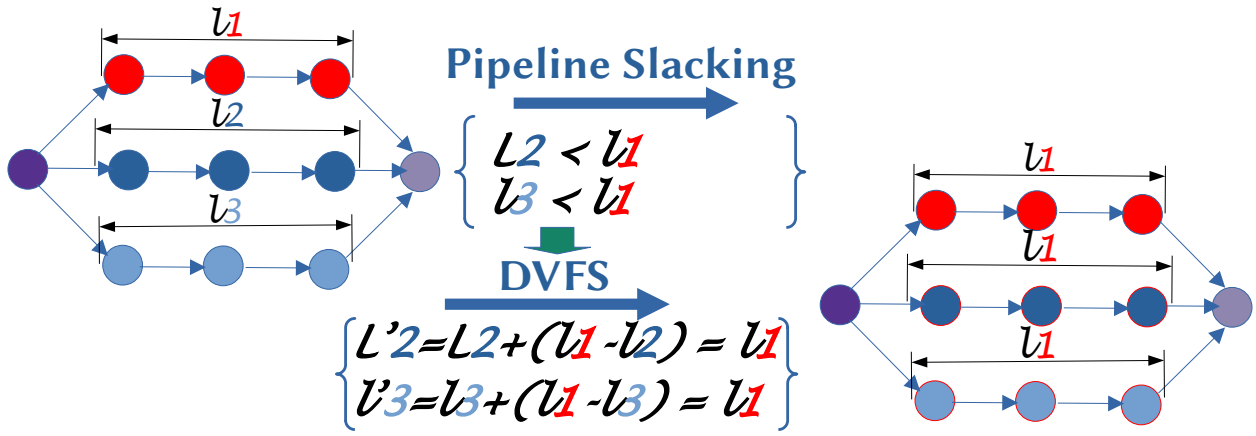


Figure 4.6: Pipelines Slacking

It combines on the one hand the IRITR evaluation along with the trade-off function, and in the second hand, some structure-based techniques presented in section 4.3.

Moreover, an improvement of the trade-off function is made by adding weight to the two trade-off factors of the function. Those weights are determined according to the structure of the workflow.

4.4.1 VM Selection

Once the workflow tasks have been ordered according to their EST, they are ready to be mapped onto the available VMs, one after the other.

In this algorithm, the trade-off factors are weighted unlike in CTTWSDP algorithm presented in chapter 3.

Given a task t_i and a VM vm_p, the trade-off between cost and time is handled according to the trade-off function as follows.

$$CTTF_i^p = costW \times Cost_i^p + timeW \times Time_i^p; \tag{4.12}$$

where $Cost_i^p$ is the cost part and $Time_i^p$ the time part, and they are evaluated according to equations (2.16) and (2.17) respectively (as presented in the section 2.2.4). The values of $costW$ and $timeW$ are given respectively by the equations (4.13) and (4.14).

$$costW = \frac{\sum_{t_i \in WT} \overline{TT(i,j)}}{\sum_{t_i \in WT} \{\overline{ET(i,k)} + \overline{TT(i,j)}\}}, t_j \in Succ(t_i); k \in IRITR \tag{4.13}$$

$$timeW = \frac{\sum_{t_i \in WT} \overline{ET(i,k)}}{\sum_{t_i \in WT} \{\overline{ET(i,k)} + \overline{TT(i,j)}\}}, t_j \in Succ(t_i); k \in IRITR \quad (4.14)$$

Dynamic VM Provisioning algorithm

It seems unsuitable to use more VMs than the largest width of the workflow for its execution. But at the same time how many VMs is suitable for the execution of the workflow? According to the level of parallelism of the workflow due to its structure, we use the average width $AvgW_{Wf}$ (described in section 3.2.2) of the workflow as limit of the number of VMs to provision for the execution.

A new VM is provisioned in preference to the best among the available VMs, if both of the following conditions are met:

- the ready time of the best VM is greater than the Min ready time of the current task (bestReadyTime > minReadyTime$_i$);
- the number of already provision VMs < $optNbVMs$.

The pseudo-code of the dynamic VM Provisioning algorithm is depicted in Algorithm 5.

4.4.2 The SCTTEWS algorithm

The pseudo-code of the SCTTEWS algorithm is depicted in Algorithm 6.

4.5 Structure-based Multi-objective Workflow Scheduling with an Optimal instance type (SMWSO)

In this section, we present our second multi-objective heuristic, named Structure-based Multi-objective Workflow Scheduling with an Optimal instance type (SMWSO). The SMWSO algorithm aims at optimizing processing costs, makespan and energy consumption, with the awareness of the user-defined budget and deadline. It dynamically determine one suitable instance type among the available types of VMs proposed and uses VMs of that type only. It consists of five main steps that are:

- The determination of the optimal instance type and the optimal number of VMs as defined in section 4.3.1;

Algorithm 5 Dynamic VM Provisioning Algorithm

Input: t_i

Output: t_i is mapped to the suitable VM ($bestVM$)

if t_i **has already been mapped to a VM then**
  $spareB^t \leftarrow spareB^t - EC(i,p)_{added}$
  return;/*do nothing since this task has been mapped through one of its parents*/
end if
for $vm_p \in VMS$ **do**
  Calculate the $CTTF_i^p$
  $bestVM \leftarrow vm_p$ **if** vm_p **fulfills the following conditions:**
  (C1) vm_p **have the highest** $CTTF_i^p$
  (C2_1) If t_i **is a root task, the VM** vm_p **must be of instance type** $expensiveIRIT$
  (C2_2) If t_i **is not a root task, the VM** vm_p **must be of an instance type belong to** $IRITR$
end for
  /*dynamic provisioning*/
if $bestReadyTime > minReadyTime_i$ **and** $length(VMS) < optNbVMs$ **then**
  for $vmit_k \in VMIT$ **do**
    Calculate the $CTTF_i^k$
    $bestVMInstance \leftarrow vmit_k$ **if** $vmit_k$ **fulfills the following conditions:**
    (C1) vm_p **have the highest** $CTTF_i^k$
    (C2_1) If t_i **is a root task,** $vmit_k$ **must be equal to** $expensiveIRIT$
    (C2_2) If t_i **is not a root task,** $vmit_k$ **must belong to** $IRITR$
  end for
  $bestVM \leftarrow vmProvisioning(bestVMInstance)$
end if
return $bestVM$
$spareB^t \leftarrow spareB^t - EC(i,p)_{added}$

Algorithm 6 SCTTEWS Algorithm

Input: The DAG of tasks

Output: All the tasks are scheduled to their suitable VMs

Order the tasks in Asc EST **into the** $readyList$
Determine the optimal number of VMs ($optNbVMs$**) as described in Section 4.3.1**
$[cheapestIRIT, expensiveIRIT] \leftarrow IRITR$ **Evaluation**
for $t_i \in readyList$ **do**
  Map t_i **to the suitable VM according to the VM selection of the Section 4.4.1**
  /*If the number of the already provision VMs is equal to $optNbVMs$, we provision no more and we chose the best among the available VMs*/
  Apply entry task duplication over t_i **if needed as described in Section 4.3.3**
  Apply pipeline merging and slacking over t_i **if needed as described in Section 4.3.4**
end for

- The task prioritization: the workflow tasks are ordered according the their descendant ranku. The ranku is defined in section 4.3.2;
- The VM selection and reuse;
- The entry task duplication as defined in section 4.3.3;
- The parallels pipelines merging and slacking as defined in section 4.3.4.

4.5.1 Determination of the optimal Instance type

It is established that instances heterogeneity can easily leads to more energy wastage, due to the workload unbalance of instances. For example, Stavrinides and Karatza [107] studied the impact of the workload and their results reveal that the workload variability has a significant impact on the energy consumption of the system. We also investigated and found that if no careful VM selection is made, using different instances for the execution of a workflow leads to more energy wastage than when one suitable instance is chosen.

Since the determination of our optimal number of VMs ($optNbVMs$) is relevant regardless the type of instance type (as presented in section 4.2.1), we can then chose the instance type ($optInstType$) which gives better results.

The effectiveness of this operation is highly dependent on the estimation of the makespan. Our makespan estimation when using only VMs of instance type $vmit_k$ (denoted as $estimateM_G^k$) is made according to equation (4.15). An illustration of that estimation is given by Figure 4.7.

$$estimateM_G^k = \frac{\sum_{t_i \in WT}\{ET(i,k) + \max_{t_j \in Succ(t_i)}\{TT(i,j)\}\}}{optNbVMs} \tag{4.15}$$

$TT(i,j)$ is the transfer time of data from task t_i to task t_j (defined in section 2.1.2).

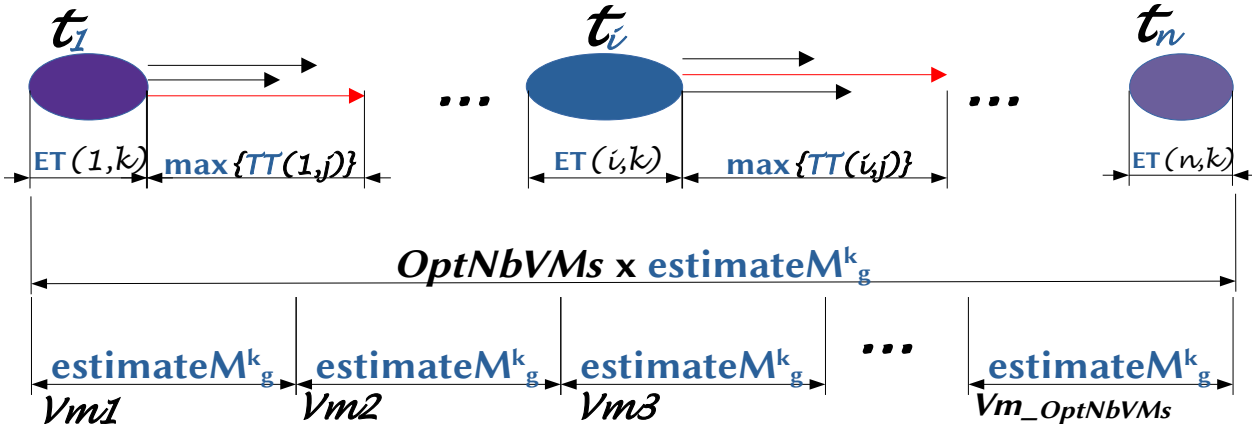


Figure 4.7: Makespan estimation when using an unique instance type $vmit_k$

Since their is a parallelism according to the workflow structure, $optInstType$ is chosen as the fastest instance type (k) which has an estimated makespan that respects the deadline with an execution cost lest than a slice of the budget corresponding to a path ($B/optNbVMs$). That means, the instance k which respects the conditions (4.16) and (4.17).

$$estimateM_G^k \leq \delta \tag{4.16}$$

$$\lceil estimateM_G^k/\tau \rceil \times c_k \leq (B/optNbVMs) \tag{4.17}$$

That is the fastest instance type which can respect both the deadline and the budget of the user, according to our makespan estimation and based on the optimal number of VMs ($optNbVMs$).

4.5.2 VM Selection and reuse

A task t_i can be mapped to a VM during the Entry Task Duplication Policy phase (see Section 4.3.3) or during the Pipeline Merging and Slacking phase (see Section 4.3.4). Therefore, when a task is selected due to its priority, it is just ignored in this phase, and then executed to the already mapped VM at due time.

When a task t_i is not yet mapped to a VM, the VM selection strategy is a modified version of the one used in HEFT [82]; ie the VM with the smallest actual finish time determined by finding the first idle time slot capable of holding the task. We determine among the already provisioned VMs, the one having the smallest actual finish time. If the corresponding start time is late on the earliest start time of the task, we proceed to the provisioning of a new VM (of type $optInstType$), taking into account the supply time. If the number of already provision VMs is up-to $optNbVMs$, we use the VM having the smallest AFT among the available VMs.

4.5.3 The SMWSO algorithm

The SMWSO algorithm uses homogeneous instances according to the determination of the optimal instance type $optInstType$.

The pseudo-code of SMWSO is depicted in Algorithm 7.

Algorithm 7 SMWSO Algorithm

Input: The DAG of tasks

Output: All the tasks are scheduled to their suitable VMs

1: **Starting from the t_{exit}, compute $rank_u$ for all tasks by using equation (4.10)**
2: **Sort the tasks list ($readyList$) in decreasing order of $rank_u$**
3: **Determine the optimal number of VMs ($optNbVMs$) and the optimal instance type ($optInstType$) as described in Section 4.3.1**
4: **for** $t_i \in readyList$ **do**
5: **Map t_i to the suitable VM according to the VM selection of the Section 4.5.2**
/*If the number of the already provision VMs is up-to $optNbVMs$, we provision no more and we use the VM having the smallest AFT*/
6: **Apply entry task duplication over t_i if needed as described in Section 4.3.3**
7: **Apply pipeline merging and slacking over t_i if needed as described in Section 4.3.4**
8: **end for**

4.6 Structure-based Multi-objective Workflow Scheduling with Heterogeneous instance types (SMWSH)

In this section, we present our third multi-objective heuristic, named Structure-based Multi-objective Workflow Scheduling with Heterogeneous instance types (SMWSH). Unlike SMWSO, SMWSH uses heterogeneous instance types in the scheduling process. SMWSH also consists of six main steps that are:

- The determination of the optimal instance type and the optimal number of VMs as defined in section 4.3.1;
- The task prioritization: the workflow tasks are ordered according the their descendant ranku. The ranku is defined in section 4.3.2;
- The deadline distribution;
- The VM selection and reuse;
- The entry task duplication as defined in section 4.3.3;
- The parallels pipelines merging and slacking as defined in section 4.3.4,

4.6.1 Deadline distribution

Since $minM_G$ is the minimum possible makespan of the workflow G, we can assume that the user defined deadline is always greater than $minM_G$ ($\delta \geq minM_G$). Let be $EST^{opt}(t_i)$ (respectively $EFT^{opt}(t_i)$) the Earliest Start Time (respectively Earliest Finish Time) of task t_i when executed on $optInstType$. We define the sub-deadline δ_i of each task t_i as follow:

$$\delta_i^{'} = \frac{EFT^{opt}(t_i) \times \delta}{minM_G}, \tag{4.18}$$

$$\delta_i = \delta_i^{'} + spare\delta_i, \tag{4.19}$$

where $spare\delta_i$ is obtain by distributing the eventual spare time ($\delta - \max_{t_i \in WT}\{\delta_i^{'}\}$) to all the task proportionally to their length compared to that of the CP.

4.6.2 VM Selection and reuse

A task t_i can be mapped to a VM during the Entry Task Duplication Policy phase (see Section 4.3.3) or during the Pipeline Merging and Slacking phase (see Section 4.3.4). Therefore, when a task is selected due to its priority, it is just ignored in this phase, and then executed to the already mapped VM at due time.

The first provisioned VM is of instance type $optInstType$. When a task t_i is not yet mapped to a VM, the VM selection strategy is a modified version of the one used in HEFT [82]; ie the VM with the smallest actual finish time determined by finding the first idle time slot capable of holding the task. We determine among the already provisioned VMs, the one having the smallest actual finish time under the sub-deadline of the task (δ_i). If such VM is not found, or the corresponding start time is late on the earliest start time of the task, we proceed to the provisioning of a new VM, taking into account the supply time. The instance type used for the provisioning is determined in the same manner than the VM; ie the instance that can end faster under the sub-deadline (δ_i). If the number of already provision VMs is up-to $optNbVMs$, we use the VM having the smallest EFT among the available VMs.

4.6.3 The SMWSH algorithm

The SMWSH algorithm begins with a VM of instance type $optInstType$. But unlike SMWSO, SMWSH handles VM selection, Entry Task Duplication Policy and Pipeline Merging and Slacking as follow:

1. *VM selection*: if the corresponding start time of the VM having the best EFT is late on the earliest start time of the task, it determines the fastest among the VMs instance that can execute the task in the deadline;

2. *Entry Task Duplication Policy*: in the sub-step 3c of the task duplication policy, since the VMs are heterogeneous, it determines an instance type that can fulfills the condition

of equation (4.11) and provision a VM of that type.

3. *Pipeline Merging and Slacking*: here also because the VMs are heterogeneous, the spare time slacking takes into account the speed of the related VMs;

The pseudo-code of SMWSH is depicted in Algorithm 8.

Algorithm 8 SMWSH Algorithm

Input: The DAG of tasks
Output: All the tasks are scheduled to their suitable VMs

1: **Starting from the t_{exit}, compute $rank_u$ for all tasks by using equation (4.10)**
2: **Sort the tasks list ($readyList$) in decreasing order of $rank_u$**
3: **Determine the optimal number of VMs ($optNbVMs$) and the optimal instance type ($optInstType$) as described in Section 4.3.1**
4: **Provision one VM of type $optInstType$**
5: **for $t_i \in readyList$ do**
6: **Map t_i to the suitable VM according to the VM selection of the Section 4.6.2**
/*If the number of the already provision VMs is up-to $optNbVMs$, we provision no more and we use the VM having the smallest AFT*/
7: **Apply entry task duplication over t_i if needed as described in Section 4.3.3**
8: **Apply pipeline merging and slacking over t_i if needed as described in Section 4.3.4**
9: **end for**

4.7 The Time Complexity of the studied algorithms

For the same raisin than in the case of CTTWS, to determine the time complexity of both SMWSO and SMWSH algorithms, the phases that must be considered are still the same: task selection and VM selection. For the task selection, given a workflow containing n tasks, we need $O(n^2)$ time for the determination of their Ranku. Sorting the tasks takes $O(nlogn)$ time complexity, hence the overall complexity for the task selection is $O(n^2)$. And for the mapping, to select the suitable VM of each ready task all the VMs should be examined. Which gives a time complexity of $O(n \times P)$ for VM selection, where P is the number of VMs. That gives a time complexity of both SMWSO and SMWSH algorithms is $O(n \times (n + P))$.

However, since the number of VMs is limited to the optimal number of VMs, we have $P = optNbVMs < n$. Therefore SMWSO and SMWSH algorithms has a polynomial time complexity of $O(n^2)$.

The time complexity of DCCP and PPDPS algorithms are presented respectively in [87] and [45], and are respectively equal to $O(n^2 \times P)$ and $O(n^2)$.

The SCTTEWS algorithm has the same time complexity than the CTTWSDP algorithm which is equal to $O(n^2)$.

The REEWS algorithm also has a time complexity order of $O(n^2)$, by using similar processes as the PPDPS algorithm in addition to the DVFS technique which does not upgrade its complexity.

The determination of time complexity of DCCP and PPDPS algorithms have been resolved in chapter 3 and are respectively $O(n^3)$ and $O(n^2)$.

A complexity comparison between the studied algorithms is summarised in Table 4.2.

Table 4.2: Complexity comparison. *Where n is the number of tasks of the workflow, K the number of used instances type and P the number of provisioned VMs instance.*

Algorithms	**DCCP**	**PPDPS**	**REEWS**	**SMWSO**	**SMWSH**	**SCTTEWS**
Complexity	$O(n^3)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$

4.8 Performance evaluation

In this section, we present the experiment's setup and analyze the simulation results.

We have used the Pegasus workflow generator [5] during experimentations to create the structure of the five real-world scientific workflows (Montage, CyberShake, Epigenomics, SIPHT, and LIGO), in different workload (the number of tasks of the workflow): 50, 100, 200, 500 and 1000 tasks.

To evaluate the performances of our three heuristics, we have implemented them as well as a state-of-the-art heuristic algorithm [21] called Reliability and Energy Efficient Workflow scheduling (REEWS). REEWS aims at minimizing energy consumption and maximizing the reliability of the system in the respect of the user-specified deadline. Unlike our proposals, the REEWS algorithm relies on static provisioning of VMs. It has been difficult for us to find a single workflow scheduling heuristic, aiming at minimizing energy consumption which uses a dynamic VMs provisioning strategy. Therefore, in order to compare our three algorithms against REEWS, we have designed their static VMs provisioning versions. We have implemented our three heuristic (dynamics and statics versions) as well as REEWS algorithm [21]. The choice of REEWS is due to the fact that it uses a (clustering) technique of determination of number of VMs to use, and the DVFS. The simulations have been done in CloudSim [99].

Also, to examine if there is a drawback in our three multi-objective algorithms in terms of success rate, we have compared them to our bi-objective algorithm Cost-Time Trade-off efficient Workflow Scheduling with Dynamic provisioning (CTTWSDP) through ANOVA test. In fact, in the precedent chapter, we have established that CTTWSDP is at least as efficient in terms of success rate than the three bi-objectives state-of-the-art heuristic algorithms considered in our work (see section 3.4.6).

4.8.1 Experiment Setup

For the simulations we consider the system as a single data center having ten different instance types that are based on the US-east (Ohio) Amazon region [95], collected in July 2019, and which the characteristics are presented in Table 4.3. The last two columns concerning the power were taken from the CloudSim framework [99] modified by Guerout et *al.* [24].

Table 4.3: Instance types based on Amazon EC2

N^o	Type	vCPU	Memory(GB)	Cost($)/Hour	Power (W)	
					Min	Max
0	m3.medium	1	3.75	0.067	140	228
1	m4.large	2	8	0.10	146	238
2	m4.xlarge	4	16	0.20	153	249
3	m4.2xlarge	8	32	0.40	159	260
4	m4.4xlarge	16	64	0.80	167	272
5	m5.8xlarge	32	128	1.536	174	282
6	m4.10xlarge	40	160	2.00	182	294
7	m5.12xlarge	48	192	2.304	188	305
8	m4.16xlarge	64	256	3.20	196	316
9	m5.24xlarge	96	384	4.608	204	330

We have configured the simulation environment as follows. The bandwidth between instances is fixed to 20 MBps, the value of the vCPU of each instance is considered as its processing capacity in Million Instruction Per Second (MIPS) as in [41], and the charging model is hourly based. For the dynamic provisioning of VMs, the provisioning delay of each VM was set to 100 s based on the study by Mao et *al.* [103]. The virtualization system used is Xen. In the case of experiment with static provisioning, we have created 10000 VMs such that the number of VMs per instance type is the same. Finally, we suppose that the DVFS is enabled on the different resource.

4.8.2 Performance Metrics

The following metrics were employed for the evaluation of the performance of our proposed scheduling algorithms:

- Cost Ratio (CR): The CR of a scheduling algorithm is overall execution cost divided by the user-defined budget (B), and determined by equation (2.18) in chapter 2 and section 2.3.2.
- Time Ratio (TR): In a similar way, the TR metric is defined as the ratio between the overall makespan and the user-defined deadline (δ), and determined by equation (2.19) in chapter 2 and section 2.3.2.

- Success Rate (SR): The SR of a scheduling algorithm is defined as the ratio between the number of ran simulations that successfully met both deadline and budget constraints (denoted by NB success), and the total number of experiments (denoted by NB_{Exp}). It is determined by equation (2.20) in chapter 2 and section 2.3.2.

- Energy consumption: The energy in kilowatt-hours (kWh) consumed by the used VMs during the observed time period;

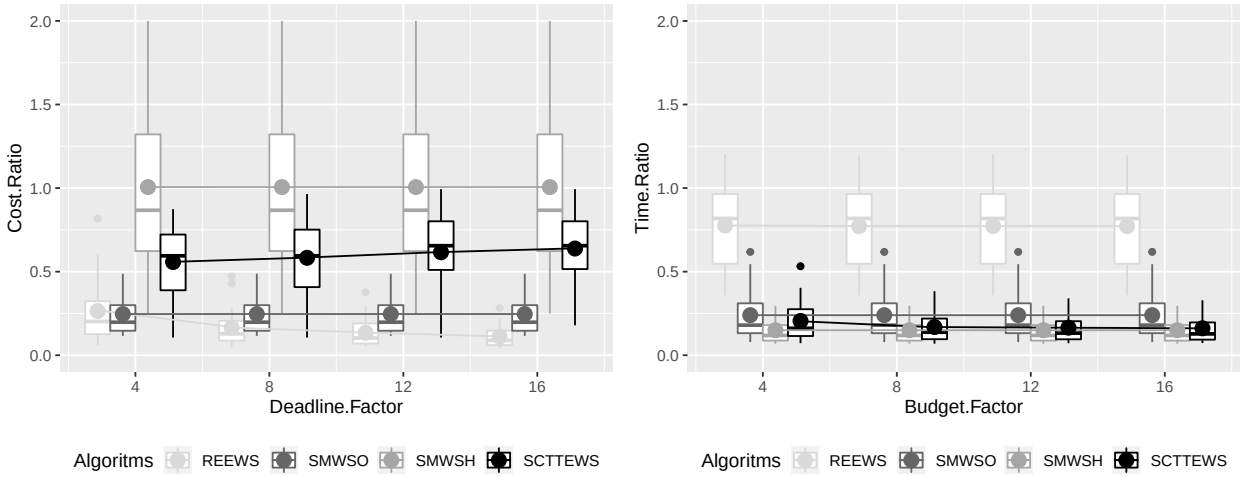
- Number of VMs: The number of VMs effectively used by each scheduling policy for the execution of the workflow. This metric is used to level of resource usage maximization. In fact using much VMs for the the execution of a workflow while it would have been possible to use less and still satisfy the user constraint is not good, since one of the objective of cloud provider been the maximization of resource utilization.

4.9 Simulation Results and Analysis with ANOVA plus Tukey-Kramer post hoc test

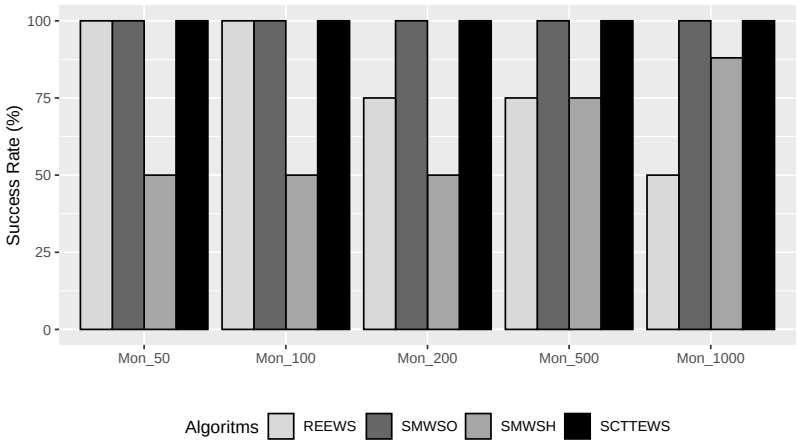
in this section we present and analyse the results of the simulation. We first analyse the results for each of the five scientific workflows used in our experiments. Afterwards, we propose a summarized analysis of the results.

Since the comparison is made against dynamic as well as static VMs provisioning algorithms, we present the results in two categories for each scientific workflow. Comparison between the dynamic provisioning algorithms, followed by the one between the static provisioning algorithms. The comparison of the dynamic version is against our proposal of chapter 3, the Cost-Time Trade-off efficient Workflow Scheduling with Dynamic provisioning (CTTWSDP) algorithm. While the static ones are compared against REEWS, which aims at minimizing energy consumption and maximizing the reliability in the respect of the user-defined deadline.

The results are presented via diverse graphs, which show the performance of the different algorithms in terms of CR, TR, SR, and energy consumption. However, in order to do objectives analysis of the results of have conducted statistical tests (ANOVA with Tukey-Kramer post hoc test). Since the energy consumption is highly dependent from the workflow type and from the workload, the statistical tests have been conducted by workflow and workload. During the simulation, they were a variation of four budget factors (4, 8, 12, 16) and four deadline factors (4, 8, 12, 16). Therefore, for each workload of each workflow, we have 16 different experiments. The summarised statistical tests are given in section 4.9.6 for both SR and energy efficiency.



(a) Cost Efficiency (Deadline Factor vs. Cost Ratio) (b) Time Efficiency (Budget Factor vs. Time Ratio)



(c) Success rate (%) compared to REEWS for MONTAGE workflow.

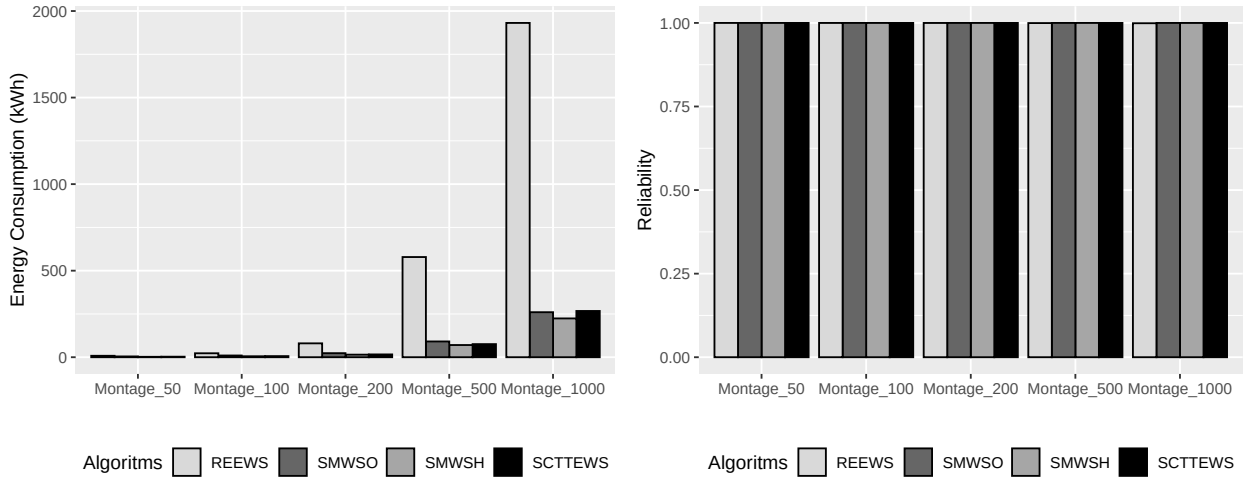
Figure 4.8: Cost efficiency, time efficiency, and success rate (%) of SCTTEWS, SMWSO and SMWSH vs REEWS for MONTAGE workflow.

4.9.1 Performance for MONTAGE workflow

Figure 4.8 presents the results obtained for MONTAGE workflow by REEWS and the static provisioning version of SCTTEWS, SMWSO and SMWSH. In terms of time efficiency (see Figure 4.8b), while our three algorithms, SCTTEWS, SMWSO and SMWSH always have 100% of schedules in the deadline, REEWS has less than 75% of schedules in the deadline. However, In terms of cost efficiency (see Figure 4.8a), SMWSH has about 60% of schedules in the budget while SCTTEWS, SMWSO and REEWS have 100% of schedules in the budget.

In terms of average success rate, while SCTTEWS and SMWSO realized 100%, REEWS and SMWSH recorded respectively 80.00% and 62.50% (see Figure 4.8c).

We noticed a significant influence of the workload of MONTAGE workflow over REEWS. When the number of tasks increases, the performance of REEWS decreases. This influence is found in SMWSH, but in reverse. When the number of tasks increases, the performance of



(a) Energy consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for MONTAGE workflow.

(b) Reliability of SCTTEWS, SMWSO and SMWSH vs REEWS for MONTAGE workflow.

Figure 4.9: Energy consumption and Reliability of SCTTEWS, SMWSO and SMWSH vs REEWS for MONTAGE workflow.

SMWSH also increases.

From Figures 4.9a and 4.9b, we observe that the energy consumption of REEWS for MONTAGE workflow is largely greater than the ones of SCTTEWS, SMWSO, and SMWSH, as the number of tasks increases. In terms of reliability, all the four algorithms have a value closer to one, which means that they produce the highest reliability. That increase in energy consumption observed on REEWS is traceable to the increase of deadline missed due to the workload. In fact, Figure 4.8c reveals a decrease of the success rate of REEWS due to the workload, and Figures 4.8a and 4.8b reveal that REEWS only fails because of deadline violation.

The statistical tests give an evidence that their is a significant different between the energy consumption of the different algorithms. In Table 4.4, all the ANOVA tests lead to the rejection of the null hypothesis (since all the p-value of the tests are smaller than 0.05). The five Tukey-Kramer post hoc tests (for the five workload of MONTAGE), which compare the mean of energy consumption between each pairwise combination of algorithms show that REEWS is always the less energy-efficient. For the workloads 50 and 100, SMWSH and SCTTEWS are the most energy-efficient, followed by SMWSO. Whereas, for the workloads 200, 500, and 1000, SMWSH, SCTTEWS, and SMWSO are similarly more energy-efficient than REEWS. That means, our proposals are more energy-efficient than REEWS for MONTAGE workflow no matter the workload.

Table 4.4: ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for MONTAGE 50, 100, 200, 500 and 10000

ANOVA input summary				
Group	***Count***	***Sum***	***Average***	***Variance***
SMWSO	16	72.16	4.51	8.41E-31
SCTTEWS	16	45.2	2.82	0.26
SMWSH	16	36.16	2.26	8.41E-31
REEWS	16	131.88	8.24	2.72

ANOVA test result						
Source of Variation	***SS***	***df***	***MS***	***F stat***	***P-value***	***F crit***
Between Groups	349.15	3	116.39	156.30	**2.59E-28**	2.76
Within Groups	44.68	60	0.74			
Total	393.85	63				

Tukey-Kramer pairwise between the four algorithms				
Comparison	***Diff***	***Abs. Diff***	***Q crit***	***Is Sig?***
SMWSO vs SCTTEWS	1.68	1.68	0.81	**YES**
SMWSO vs SMWSH	2.25	2.25	0.81	**YES**
SMWSO vs REEWS	-3.73	3.73	0.81	**YES**
SCTTEWS vs SMWSH	0.56	0.56	0.81	NO
SCTTEWS vs REEWS	-5.42	5.42	0.81	**YES**
SMWSH vs REEWS	-5.98	5.98	0.81	**YES**

Algorithm Ranking in terms of Energy saving : SMWSO $\rightarrowtail 3$ | ***SCTTEWS*** $\rightarrowtail 1$ | ***SMWSH*** $\rightarrowtail 1$ | ***REEWS*** $\rightarrowtail 4$

(a) ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for **MONTAGE 50**

ANOVA input summary				
Group	***Count***	***Sum***	***Average***	***Variance***
SMWSO	16	159.04	9.94	0
SCTTEWS	16	95.38	5.96	0.07
SMWSH	16	88.16	5.51	8.41E-31
REEWS	16	363.92	22.745	40.97

ANOVA test result						
Source of Variation	***SS***	***df***	***MS***	***F stat***	***P-value***	***F crit***
Between Groups	3113.47	3	1037.83	101.14	1.96E-23	2.76
Within Groups	615.72	60	10.26			
Total	3729.18	63				

Tukey-Kramer pairwise between the four algorithms				
Comparison	***Diff***	***Abs. Diff***	***Q crit***	***Is Sig?***
SMWSO vs SCTTEWS	3.98	3.98	2.99	**YES**
SMWSO vs SMWSH	4.43	4.43	2.99	**YES**
SMWSO vs REEWS	-12.80	12.80	2.99	**YES**
SCTTEWS vs SMWSH	0.45	0.45	2.99	NO
SCTTEWS vs REEWS	-16.78	16.78	2.99	**YES**
SMWSH vs REEWS	-17.23	17.23	2.99	**YES**

Algorithm Ranking in terms of Energy saving : SMWSO $\rightarrowtail 3$ | ***SCTTEWS*** $\rightarrowtail 1$ | ***SMWSH*** $\rightarrowtail 1$ | ***REEWS*** $\rightarrowtail 4$

(b) ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for **MONTAGE 100**

ANOVA input summary				
Group	***Count***	***Sum***	***Average***	***Variance***
SMWSO	16	373.12	23.32	1.35E-29
SCTTEWS	16	244.65	15.29	0.07
SMWSH	16	237.92	14.87	1.35E-29
REEWS	16	1282.96	80.18	497.17

ANOVA test result						
Source of Variation	***SS***	***df***	***MS***	***F stat***	***P-value***	***F crit***
Between Groups	47388.03	3	15796.01	127.07	5.91E-26	2.76
Within Groups	7458.61	60	124.31			
Total	54846.64	63				

Tukey-Kramer pairwise between the four algorithms				
Comparison	***Diff***	***Abs. Diff***	***Q crit***	***Is Sig?***
SMWSO vs SCTTEWS	8.03	8.03	10.42	NO
SMWSO vs SMWSH	8.45	8.45	10.42	NO
SMWSO vs REEWS	-56.86	56.86	10.42	**YES**
SCTTEWS vs SMWSH	0.42	0.42	10.42	NO
SCTTEWS vs REEWS	-64.89	64.89	10.42	**YES**
SMWSH vs REEWS	-65.31	65.31	10.42	**YES**

Algorithm Ranking in terms of Energy saving : SMWSO $\rightarrowtail 1$ | ***SCTTEWS*** $\rightarrowtail 1$ | ***SMWSH*** $\rightarrowtail 1$ | ***REEWS*** $\rightarrowtail 4$

(c) ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for **MONTAGE 200**

ANOVA input summary				
Group	***Count***	***Sum***	***Average***	***Variance***
SMWSO	16	1456.32	91.02	0
SCTTEWS	16	1130.42	70.65	0.09
SMWSH	16	1126.02	70.38	2.5E-05
REEWS	16	9260.4	578.77	48639.91

ANOVA test result						
Source of Variation	***SS***	***df***	***MS***	***F stat***	***P-value***	***F crit***
Between Groups	3021620.40	3	1007206.8	82.83	2.62E-21	2.76
Within Groups	729600.13	60	12160.00			
Total	3751220.53	63				

Tukey-Kramer pairwise between the four algorithms				
Comparison	***Diff***	***Abs. Diff***	***Q crit***	***Is Sig?***
SMWSO vs SCTTEWS	20.37	20.37	103.02	NO
SMWSO vs SMWSH	20.64	20.64	103.02	NO
SMWSO vs REEWS	-487.75	487.75	103.02	**YES**
SCTTEWS vs SMWSH	0.275	0.275	103.02	NO
SCTTEWS vs REEWS	-508.12	508.12	103.02	**YES**
SMWSH vs REEWS	-508.40	508.40	103.02	**YES**

Algorithm Ranking in terms of Energy saving : SMWSO $\rightarrowtail 1$ | ***SCTTEWS*** $\rightarrowtail 1$ | ***SMWSH*** $\rightarrowtail 1$ | ***REEWS*** $\rightarrowtail 4$

(d) ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for **MONTAGE 500**

ANOVA input summary				
Group	***Count***	***Sum***	***Average***	***Variance***
SMWSO	16	4160.32	260.02	0
SCTTEWS	16	3994.99	249.69	2007.02
SMWSH	16	3585.49	224.09	0.13
REEWS	16	30900.25	1931.26	441719.03

ANOVA test result						
Source of Variation	***SS***	***df***	***MS***	***F stat***	***P-value***	***F crit***
Between Groups	34149038.12	3	11383012.71	102.61	1.36E-23	2.76
Within Groups	6655892.89	60	110931.55			
Total	40804931.01	63				

Tukey-Kramer pairwise between the four algorithms				
Comparison	***Diff***	***Abs. Diff***	***Q crit***	***Is Sig?***
SMWSO vs SCTTEWS	10.33	10.33	311.17	NO
SMWSO vs SMWSH	35.93	35.93	311.17	NO
SMWSO vs REEWS	-1671.24	1671.24	311.17	**YES**
SCTTEWS vs SMWSH	25.59	25.59	311.17	NO
SCTTEWS vs REEWS	-1681.58	1681.58	311.17	**YES**
SMWSH vs REEWS	-1707.17	1707.17	311.17	**YES**

Algorithm Ranking in terms of Energy saving : SMWSO $\rightarrowtail 1$ | ***SCTTEWS*** $\rightarrowtail 1$ | ***SMWSH*** $\rightarrowtail 1$ | ***REEWS*** $\rightarrowtail 4$

(e) ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for **MONTAGE 1000**

4.9.2 Performance for CYBERSHAKE workflow

For CYBERSHAKE workflow, all the four algorithms SCTTEWS, SMWSO, SMWSH, and REEWS have a good time efficiency (see Figure 4.10b). However, in terms of cost efficiency only REEWS realises 100% of schedules in the budget, whereas SCTTEWS, SMWSO and SMWSH have 25% of schedules out of the budget (see Figure 4.10b).

In terms of average success rate, REEWS recorded 100% whereas SCTTEWS, SMWSO and SMWSH realized respectively 95.00%, 78.75% and 85.00%.

REEWS is more energy-efficient than SCTTEWS, SMWSO, and SMWSH for CYBERSHAKE workflow (see Figure 4.11a). In terms of reliability, all the four algorithms have the highest reliability.

In the case of CYBERSHAKE also, the statistical tests give an evidence that their is a significant different between the energy consumption of the different algorithms. All the ANOVA tests still lead to the rejection of the null hypothesis (see Table A.1, all the p-value of the tests are smaller than 0.05). The Tukey-Kramer post hoc tests (for the five workload of CYBERSHAKE) reveal that SMWSO is always the less energy-efficient. For the workload 50 REEWS and SMWSH are the most energy-efficient, followed by SCTTEWS. For the workloads 100, 200, and 500, REEWS, SMWSH, and SCTTEWS are similarly more energy-efficient than SMWSO. Finaly in the case of the workloads 1000, SMWSH is the most energy-efficient, followed by SMWSO and SCTTEWS.

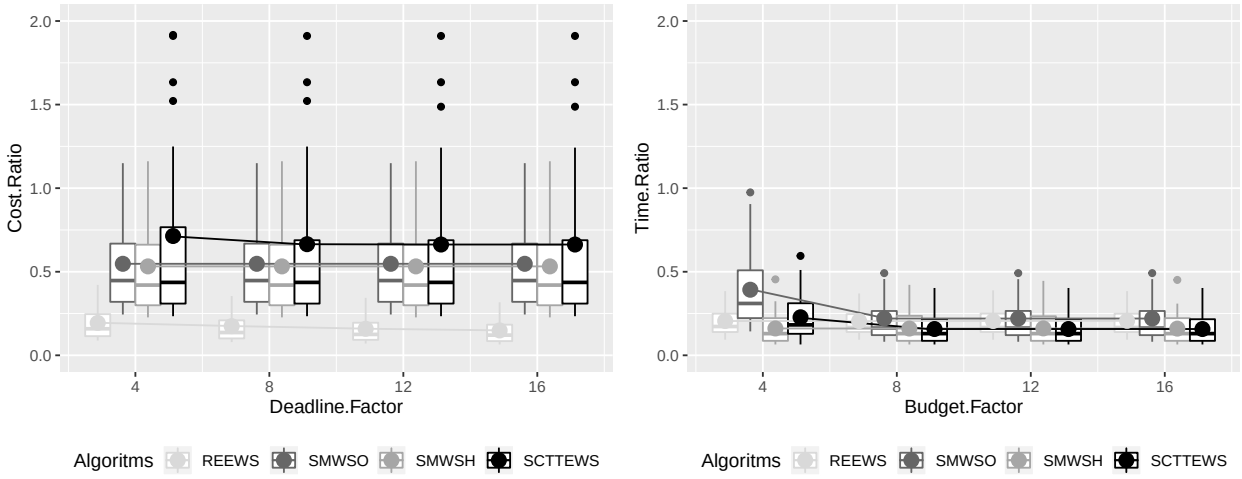
4.9.3 Performance for EPIGENOMICS workflow

For EPIGENOMICS workflow, all the four algorithms SCTTEWS, SMWSO, SMWSH, and REEWS have 100% of schedules in the budget (see Figure 4.10a). However, in terms of cost efficiency only SMWSO and REEWS realise 100% of schedules in the deadline, whereas SCTTEWS and SMWSH have few schedules out of the deadline (see Figure 4.10b).

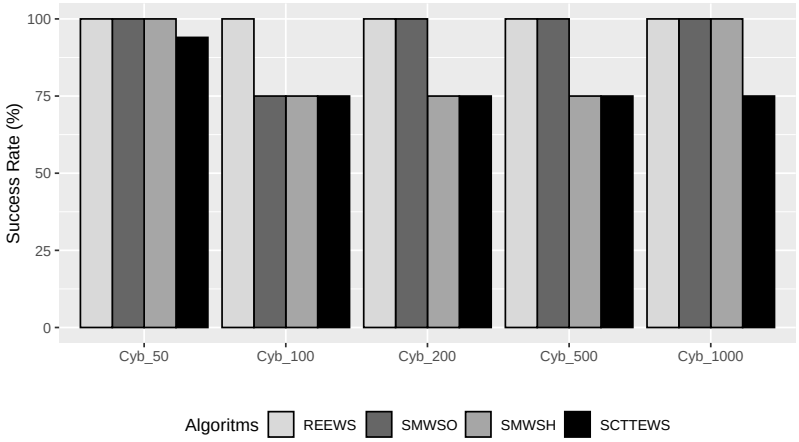
In terms of average success rate, SMWSO and REEWS recorded 100% whereas SCTTEWS and SMWSH realized respectively 90.00% and 95.00%.

SMWSO is more energy-efficient than REEWS, SCTTEWS, and SMWSH for EPIGENOMICS workflow (see Figure 4.13a). In terms of reliability, SMWSO, SCTTEWS, and SMWSH are highly reliable, and slightly more reliable than REEWS. We notice that for the workload of 1000 tasks, SCTTEWS and SMWSH lead to greater energy consumption compared to SMWSO and REEWS. This was not the case for the smaller workload.

The statistical tests for in the case of EPIGENOMICS also give an evidence that their is a significant different between the energy consumption of the different algorithms. All the

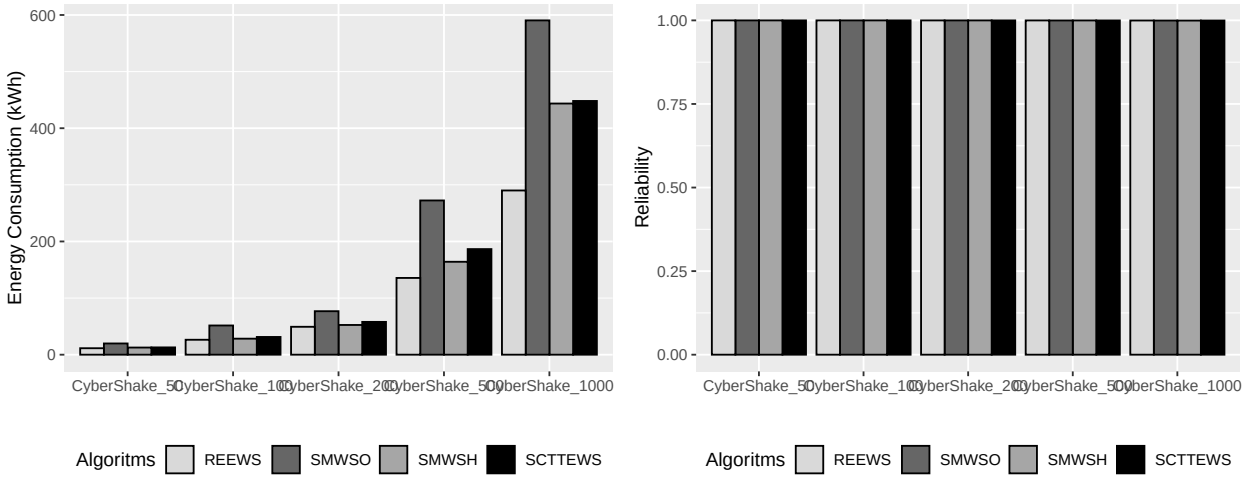


(a) Cost Efficiency (Deadline Factor vs. Cost Ratio) (b) Time Efficiency (Budget Factor vs. Time Ratio)



(c) Success rate (%) compared to REEWS for CYBERSHAKE workflow.

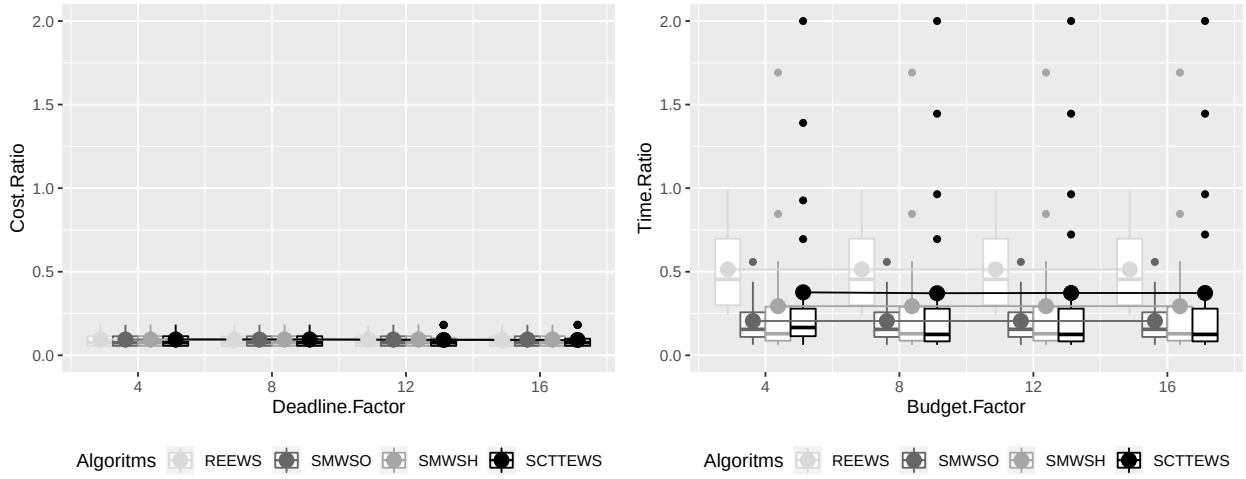
Figure 4.10: Cost efficiency, time efficiency, and success rate (%) of SCTTEWS, SMWSO and SMWSH vs REEWS for CYBERSHAKE workflow.



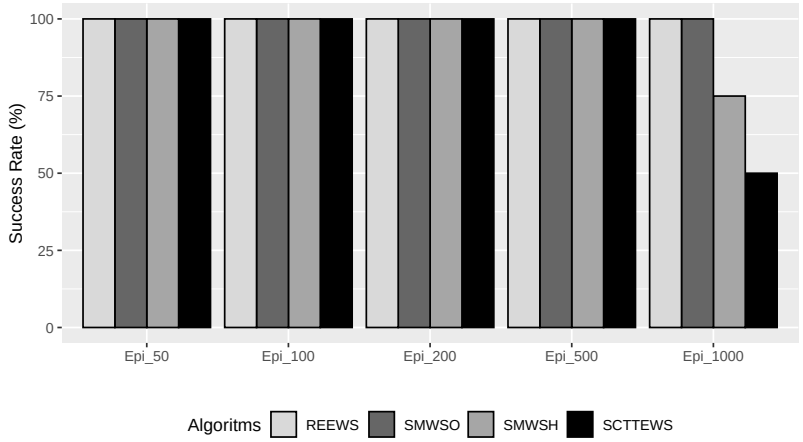
(a) Energy consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for CYBERSHAKE workflow.

(b) Reliability of SCTTEWS, SMWSO and SMWSH vs REEWS for CYBERSHAKE workflow.

Figure 4.11: Energy consumption and Reliability of SCTTEWS, SMWSO and SMWSH vs REEWS for CYBERSHAKE workflow.

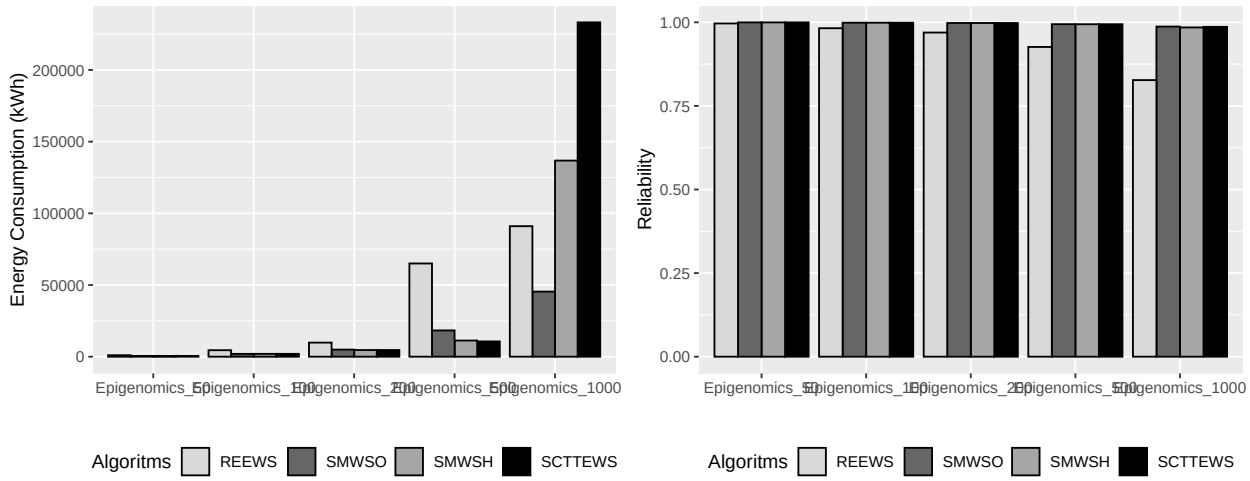


(a) Cost Efficiency (Deadline Factor vs. Cost Ratio) (b) Time Efficiency (Budget Factor vs. Time Ratio)



(c) Success rate (%) compared to REEWS for EPIGENOMICS workflow.

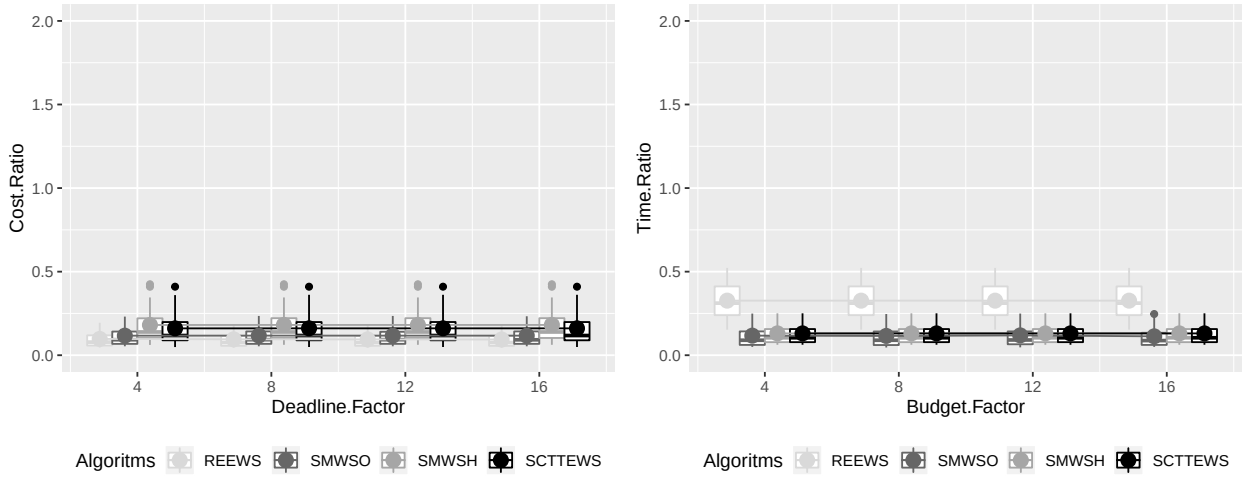
Figure 4.12: Cost efficiency, time efficiency, and success rate (%) of SCTTEWS, SMWSO and SMWSH vs REEWS for EPIGENOMICS workflow.



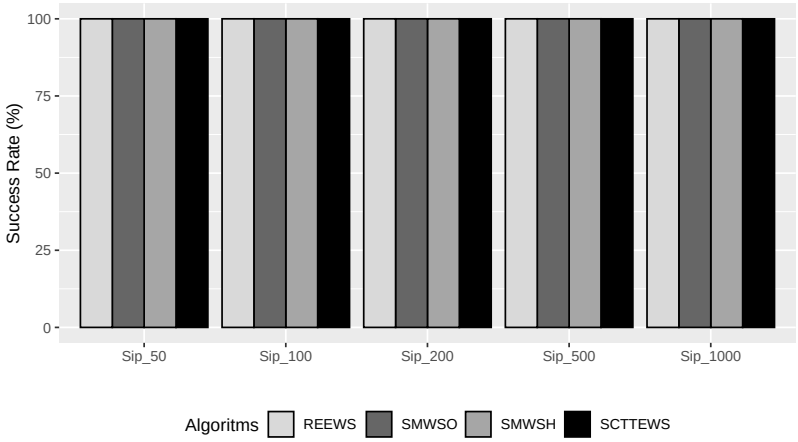
(a) Energy consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for EPIGENOMICS workflow.

(b) Reliability of SCTTEWS, SMWSO and SMWSH vs REEWS for EPIGENOMICS workflow.

Figure 4.13: Energy consumption and Reliability of SCTTEWS, SMWSO and SMWSH vs REEWS for EPIGENOMICS workflow.



(a) Cost Efficiency (Deadline Factor vs. Cost Ratio) (b) Time Efficiency (Budget Factor vs. Time Ratio)



(c) Success rate (%) compared to REEWS for SIPHT workflow.

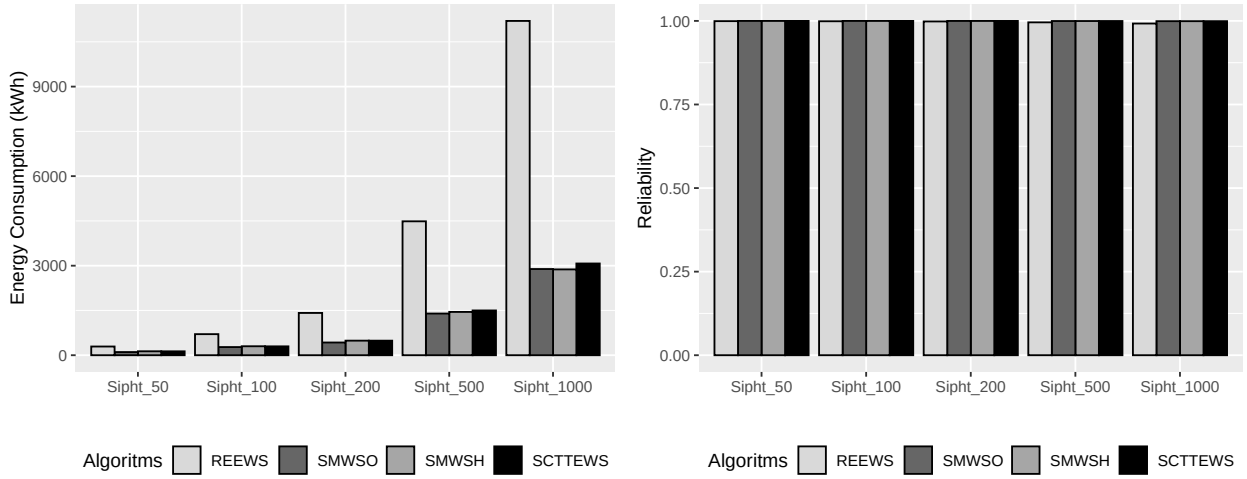
Figure 4.14: Cost efficiency, time efficiency, and success rate (%) of SCTTEWS, SMWSO and SMWSH vs REEWS for SIPHT workflow.

ANOVA tests still lead to the rejection of the null hypothesis (see Table A.2, all the p-value of the tests are smaller than 0.05). The Tukey-Kramer post hoc tests (for the five workload of EPIGENOMICS) reveal the following. For workload 50, SMWSO and SCTTEWS are the most energy-efficient, followed by SMWSH, REEWS been the less energy-efficient. For the workloads 100, 200, and 500, SMWSO, SMWSH, and SCTTEWS are similarly more energy-efficient than REEWS. Finally, in the case of the workloads 1000, SMWSO is the most energy-efficient, followed by REEWS, afterward by SMWSH, SCTTEWS been the less energy-efficient.

4.9.4 Performance for SIPHT workflow

For SIPHT workflow, all the four algorithms SCTTEWS, SMWSO, SMWSH, and REEWS have 100% of schedules in both the budget and the deadline (see Figures 4.14a and 4.14b).

Therefore, SCTTEWS, SMWSO, SMWSH, and REEWS have 100% of average success rate.



(a) Energy consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for SIPHT workflow.

(b) Reliability of SCTTEWS, SMWSO and SMWSH vs REEWS for SIPHT workflow.

Figure 4.15: Energy consumption and Reliability of SCTTEWS, SMWSO and SMWSH vs REEWS for SIPHT workflow.

REEWS is less energy-efficient than SCTTEWS, SMWSO, and SMWSH for SIPHT workflow (see Figure 4.15a). All the four algorithms are highly reliable. The workload of SIPHT also has significant impact on the energy-efficiency of REEWS.

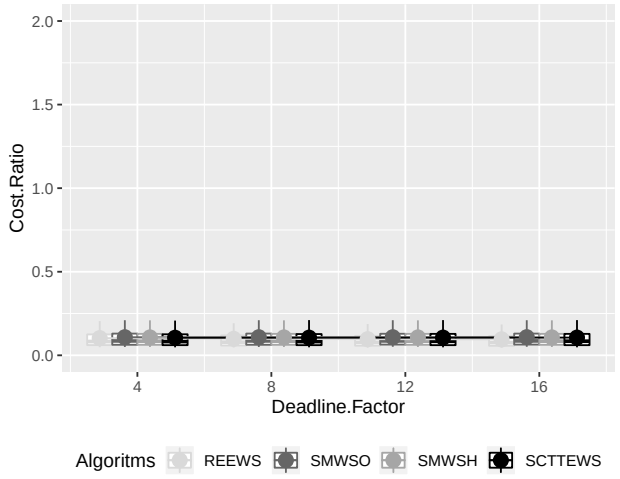
In the case of SIPHT also, the ANOVA tests proved that their is a significant different between the energy consumption of the different algorithms (see Table A.3, all the p-value of the tests are smaller than 0.05). The Tukey-Kramer post hoc tests (for the five workload of SIPHT) reveal that REEWS is always the less energy-efficient. For the workload 50 SMWSO is the most energy-efficient, followed by SCTTEWS and SMWSH. For the workloads 100, 200, 500 and 1000, SMWSO, SCTTEWS, and SMWSH are similarly more energy-efficient than REEWS.

4.9.5 Performance for LIGO workflow

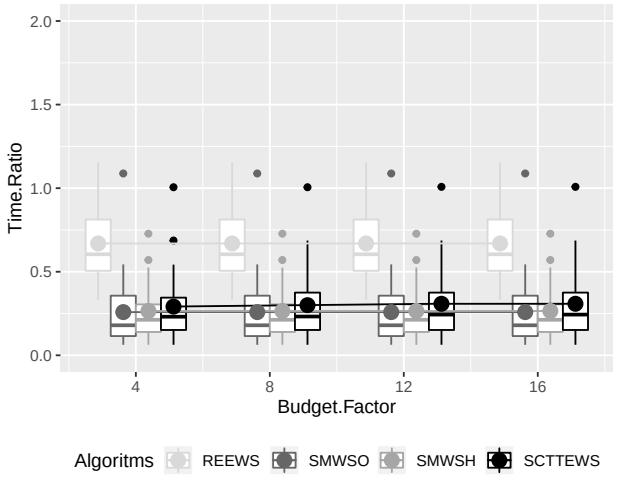
For LIGO workflow, SMWSH has 100% of schedules in both the budget and the deadline (see Figures 4.16a and 4.16b). SCTTEWS, SMWSO and REEWS have few schedules out of the deadline and 100% of schedules in the budget.

The average of success rate of SMWSH is 100%, whereas SCTTEWS, SMWSO and REEWS have of respectively 95.00%, 95.00% and 85.00%.

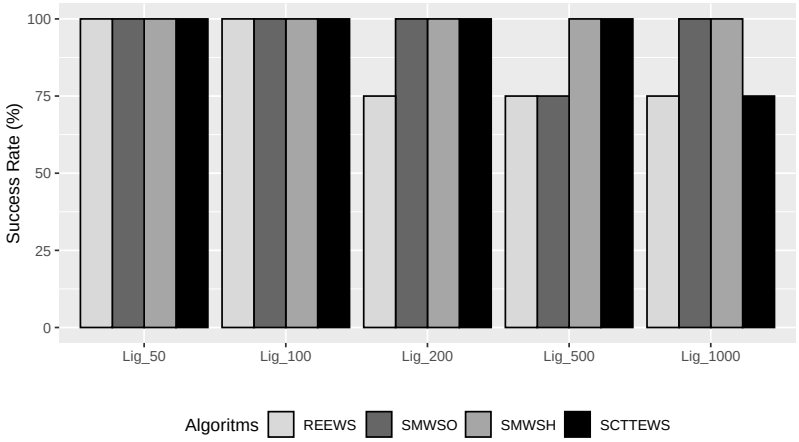
REEWS is less energy-efficient than SCTTEWS, SMWSO, and SMWSH for LIGO workflow (see Figure 4.15a). All the four algorithms are highly reliable, apart in the case of the workload 500 and 1000 that REEWS is slightly less reliable than the other. The workload of LIGO also has significant impact on the energy-efficiency of REEWS.



(a) Cost Efficiency (Deadline Factor vs. Cost Ratio)

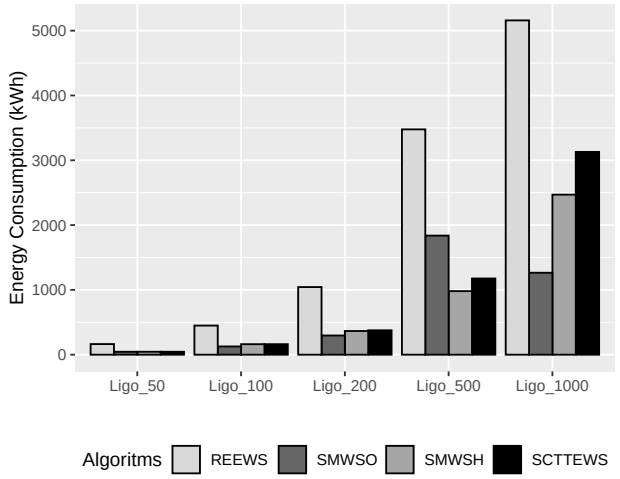


(b) Time Efficiency (Budget Factor vs. Time Ratio)

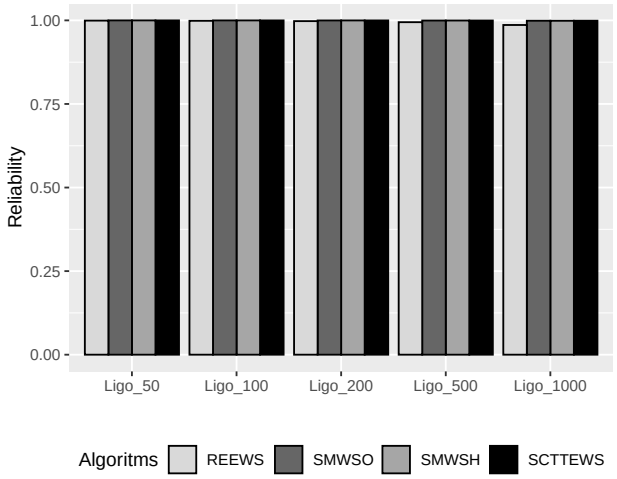


(c) Success rate (%) compared to REEWS for LIGO workflow.

Figure 4.16: Cost efficiency, time efficiency, and success rate (%) of SCTTEWS, SMWSO and SMWSH vs REEWS for LIGO workflow.



(a) Energy consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for LIGO workflow.



(b) Reliability of SCTTEWS, SMWSO and SMWSH vs REEWS for LIGO workflow.

Figure 4.17: Energy consumption and Reliability of SCTTEWS, SMWSO and SMWSH vs REEWS for LIGO workflow.

All the ANOVA tests still lead to the rejection of the null hypothesis (see Table A.4, all the p-value of the tests are smaller than 0.05), and therefore prove that their is a significant different between the energy consumption of the different algorithms. The Tukey-Kramer post hoc tests (for the five workload of LIGO) reveal that REEWS is always the less energy-efficient. For the workload 50, 100, and 200, SMWSO, SCTTEWS and SMWSH are similarly more energy-efficient than SMWSO. For the workloads 500, SCTTEWS and SMWSH are the most energy-efficient, followed by SMWSH. Whereas, for the workloads 1000, SMWSO is the most energy-efficient, followed by SCTTEWS and SMWSH.

4.9.6 Performance summary and discussions

In this subsection, we present a summary of the simulation results in terms of success rate and in terms of energy efficiency, and provide some analysis.

In terms of success rate

First of all we analyse the success rate obtained for the dynamic provisioning algorithms. Table 4.5 presents the total SR of the four dynamics algorithms CTTWSDP, SCTTEWS, SMWSO and SMWSH. CTTWSDP is our bi-objective algorithm proposed in the previous chapter.

From the Table 4.5, we see that CTTWSDP, SCTTEWS, SMWSO and SMWSH algorithms recorded respectively 94.36%, 97.00%, 99.00% and 96.00% in terms mean of SR. However, to know if that difference in terms of SR (i.e. customer satisfaction) is significant, we have conducted ANOVA test[2] [93, 104].

Table 4.6 presents the result of the ANOVA test. As we can see in Table 4.6b, we have F statistical < F critical (also p-value > 0.05). Thus, we have sufficient evidence to say that all the four algorithms CTTWSDP, SCTTEWS, SMWSO, and SMWSH have similar performance in terms of success rate. Therefore, there is no drawback incurred in our three energy-efficient algorithms in terms of success rate (i.e. customer satisfaction) while we tried to satisfied the provider.

As for the four static algorithms REEWS, SCTTEWS, SMWSO, and SMWSH, Table 4.7 presents the summary of the average success rate realized by each. REEWS, SCTTEWS, SMWSO and SMWSH algorithms recorded respectively 93.00%, 92.75%, **98.00**% and 88.50% in terms mean of SR. Even though SMWSO has higher success rate than the other with the smallest standard deviation (*2.44*), a statistical test is still needed. We did an ANOVA test to know if that difference in terms of SR (i.e. customer satisfaction) is significant.

[2]https://www.excel-easy.com/examples/anova.html

Table 4.5: Success rate summary realized by the four dynamics algorithms CTTWSDP, SCTTEWS, SMWSO and SMWSH.

Workflow	CTTWSDP	SCTTEWS	SMWSO	SMWSH
MONTAGE 50	99.58%	100%	100%	100%
MONTAGE 100	99.58%	100%	100%	100%
MONTAGE 200	100%	100%	100%	100%
MONTAGE 500	100%	100%	100%	100%
MONTAGE 1000	100%	100%	100%	100%
CYBERSHAKE 50	99.79%	100%	100%	100%
CYBERSHAKE 100	98.96%	75.00%	75.00%	75.00%
CYBERSHAKE 200	93.75%	75.00%	100%	75.00%
CYBERSHAKE 500	95.42%	100%	100%	75.00%
CYBERSHAKE 1000	83.13%	75.00%	100%	75.00%
EPIGENOMICS 50	97.92%	100%	100%	100%
EPIGENOMICS 100	92.71%	100%	100%	100%
EPIGENOMICS 200	99.17%	100%	100%	100%
EPIGENOMICS 500	100%	100%	100%	100%
EPIGENOMICS 1000	100%	100%	100%	100%
SIPHT 50	78.33%	100%	100%	100%
SIPHT 100	97.50%	100%	100%	100%
SIPHT 200	96.46%	100%	100%	100%
SIPHT 500	97.08%	100%	100%	100%
SIPHT 1000	96.46%	100%	100%	100%
LIGO 50	84.79%	100%	100%	100%
LIGO 100	85.63%	100%	100%	100%
LIGO 200	86.25%	100%	100%	100%
LIGO 500	93.96%	100%	100%	100%
LIGO 1000	82.50%	100%	100%	100%
Mean	94.36%	97.00%	**99.00**%	96.00%

Table 4.6: ANOVA test result comparing the SR of the four dynamics algorithms CTTWSDP, SCTTEWS, SMWSO and SMWSH

Group	***Count***	***Sum***	***Average***	***Variance***
CTTWSDP	25	2358.97	94.36	45.32
SCTTEWS	25	2425	97	68.75
SMWSO	25	2475	99	25
SMWSO	25	2400	96	87.50

(a) Summary of input

Source of Variation	***SS***	***df***	***MS***	***F statistical***	***P-value***	***F critical***
Between Groups	282.56	3	94.19	1.66	0.18	2.70
Within Groups	5437.76	96	56.64			
Total	5720.32	99				

(b) ANOVA test result

Table 4.7: Success rate summary realized by the four static algorithms REEWS, SCTTEWS, SMWSO and SMWSH.

Workflow	**REEWS**	**SCTTEWS**	**SMWSO**	**SMWSH**
MONTAGE 50	100%	100%	100%	50%
MONTAGE 100	100%	100%	100%	50%
MONTAGE 200	75%	100%	100%	50%
MONTAGE 500	75%	100%	100%	75%
MONTAGE 1000	50%	100%	100%	88%
CYBERSHAKE 50	100%	94%	100%	100%
CYBERSHAKE 100	100%	75%	75%	75%
CYBERSHAKE 200	100%	75%	100%	75%
CYBERSHAKE 500	100%	75%	100%	75%
CYBERSHAKE 1000	100%	75%	100%	100%
EPIGENOMICS 50	100%	100%	100%	100%
EPIGENOMICS 100	1001%	100%	100%	100%
EPIGENOMICS 200	100%	100%	100%	100%
EPIGENOMICS 500	100%	100%	100%	100%
EPIGENOMICS 1000	100%	50%	100%	75%
SIPHT 50	100%	100%	100%	100%
SIPHT 100	100%	100%	100%	100%
SIPHT 200	100%	100%	100%	100%
SIPHT 500	100%	100%	100%	100%
SIPHT 1000	100%	100%	100%	100%
LIGO 50	100%	100%	100%	100%
LIGO 100	100%	100%	100%	100%
LIGO 200	75%	100%	100%	100%
LIGO 500	75%	100%	75%	100%
LIGO 1000	75%	75%	100%	100%
Mean	93.00%	92.76%	**98.00%**	88.52%
Standard deviation	8.71	7.92	**2.44**	14.10

Table 4.8 presents the result of the ANOVA test comparing the four static algorithms. As we can see in Table 4.8b, we have F statistical $<$ F critical (also p-value $>$ 0.05). We then conclude that all the four algorithms REEWS, SCTTEWS, SMWSO and SMWSH have similar performance in terms of success rate.

The above statistical analyzes show that the four algorithms REEWS, SCTTEWS, SMWSO, and SMWSH have comparable performance in terms of success rate (customer satisfaction). Moreover, our three energy-efficient algorithms don't incur drawbacks in terms of success rate (i.e. customer satisfaction) while we tried to satisfied the provider relatively to our bi-objective algorithm presented in chapter 3.

However, we notice a drawback when we compare le performance of SMWSH in its dynamic provisioning version against its static version (see Figures 4.5 and 4.7) .

In terms of energy efficiency

Because of the high influence of the workflow type and workload over the energy consumption, the statistical tests have been conducted by workflow and workload in section 4.9. Table 4.9

Table 4.8: ANOVA test result comparing the SR of the four static algorithms REEWS, SCTTEWS, SMWSO and SMWSH

Group	***Count***	***Sum***	***Average***	***Variance***
REEWS	25	2325	93	183.33
SCTTEWS	25	2319	92.76	181.27
SMWSO	25	2450	98	47.92
SMWSH	25	2213	88.52	311.43

(a) Summary of input

Source of Variation	***SS***	***df***	***MS***	***F statistical***	***P-value***	***F critical***
Between Groups	1127.71	3	375.90	2.08	0.11	2.67
Within Groups	17374.8	96	180.99			
Total	18502.51	99				

(b) ANOVA test result

presents the summary of the energy efficiency ranking between the four algorithms REEWS, SCTTEWS, SMWSO and SMWSH. From Table 4.9, we see that the REEWS algorithm is more energy-efficient or as energy-efficient as our proposals only for the case of CYBERSHAKE (50, 100, 200, 500, and 1000). In all the other case, REEWS is always the less energy-efficient, and our proposals that are more energy-efficient:

- *For MONTAGE*: SCTTEWS and SMWSH are similarly more energy-efficient for all five workloads. SMWSO is as energy-efficient as SCTTEWS and SMWSH for MONTAGE 200, 500, and 1000, and second them for MONTAGE 50 and 100.
- *For CYBERSHAKE*: In the case of the workloads CYBERSHAKE 100, 200 and 500, REEWS, SCTTEWS, and SMWSH are similarly more energy-efficient than SMWSO. In the case of the workload CYBERSHAKE 50, REEWS and SMWSH are similarly more energy-efficient, followed by SCTTEWS, and finally SMWSO. Lastly, for the case of the workload CYBERSHAKE 1000, REEWS is more energy-efficient, followed by SCTTEWS and SMWSH, and finally SMWSO.
- *For EPIGENOMICS*: In the case of the workloads EPIGENOMICS 100, 200 and 500, SMWSO, SCTTEWS, and SMWSH are similarly more energy-efficient than REEWS. In the case of the workload EPIGENOMICS 50, SMWSO and SCTTEWS are similarly more energy-efficient, followed by SMWSH, and lastly REEWS. finally, for the case of the workload EPIGENOMICS 1000, SMWSO is more energy-efficient, followed by REEWS, then by SCTTEWS, and finally SMWSH.
- *For SIPHT*: In the case of the workloads SIPHT 100, 200, 500, and 1000, SMWSO, SCTTEWS, and SMWSH are similarly more energy-efficient than REEWS. As for the workload SIPHT 50, SMWSO is the more energy-efficient, followed by SCTTEWS and SMWSH, and lastly REEWS.

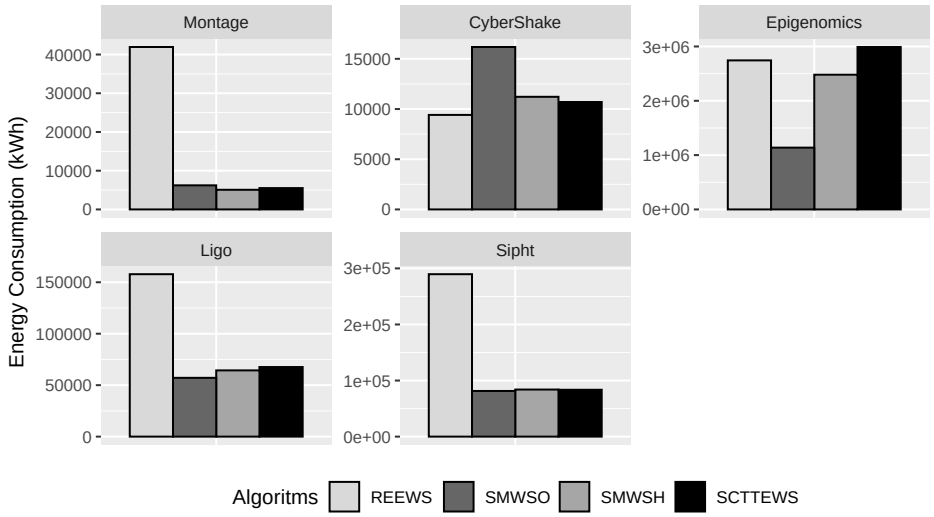
- *For LIGO*: In the case of the workloads LIGO 50, 100 and 200, SMWSO, SCTTEWS, and SMWSH are similarly more energy-efficient than REEWS. In the case of the workload LIGO 500, SCTTEWS and SMWSH are similarly more energy-efficient, followed by SMWSO, and lastly REEWS. finally, for the case of the workload LIGO 1000, SMWSO is more energy-efficient, followed by SMWSH, then by SCTTEWS, and finally REEWS.

Table 4.9: Energy efficiency ranking between the four static algorithms REEWS, SCTTEWS, SMWSO and SMWSH.

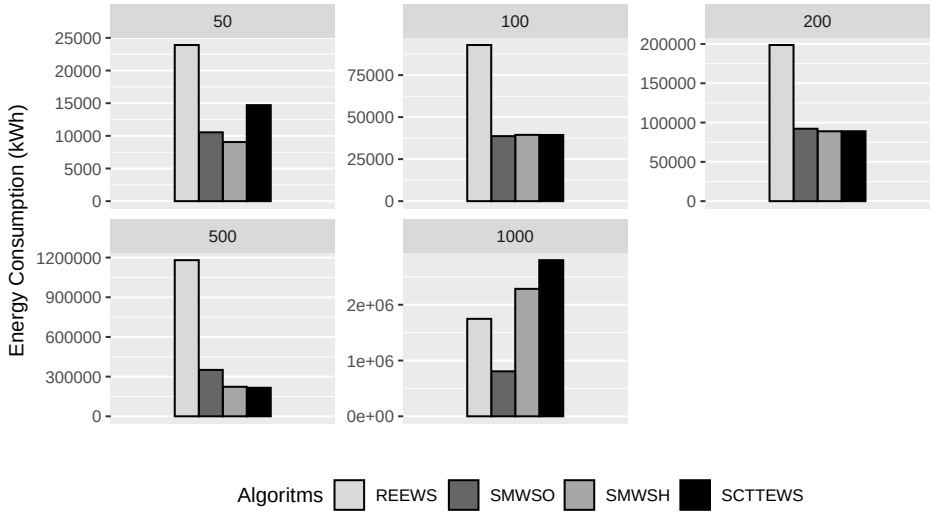
Workflow	**REEWS**	**SCTTEWS**	**SMWSO**	**SMWSH**
MONTAGE 50	4	1	3	1
MONTAGE 100	4	1	3	1
MONTAGE 200	4	1	1	1
MONTAGE 500	4	1	1	1
MONTAGE 1000	4	1	1	1
CYBERSHAKE 50	1	3	4	1
CYBERSHAKE 100	1	1	4	1
CYBERSHAKE 200	1	1	4	1
CYBERSHAKE 500	1	1	4	1
CYBERSHAKE 1000	1	2	4	2
EPIGENOMICS 50	4	1	1	3
EPIGENOMICS 100	4	1	1	1
EPIGENOMICS 200	4	1	1	1
EPIGENOMICS 500	4	1	1	1
EPIGENOMICS 1000	2	4	1	3
SIPHT 50	4	2	1	2
SIPHT 100	4	1	1	1
SIPHT 200	4	1	1	1
SIPHT 500	4	1	1	1
SIPHT 1000	4	1	1	1
LIGO 50	4	1	1	1
LIGO 100	4	1	1	1
LIGO 200	4	1	1	1
LIGO 500	4	1	3	1
LIGO 1000	4	2	1	2

The results prove the significant improvement of our proposal in terms of energy-saving, mainly the SMWSO algorithm. We have advocated that:

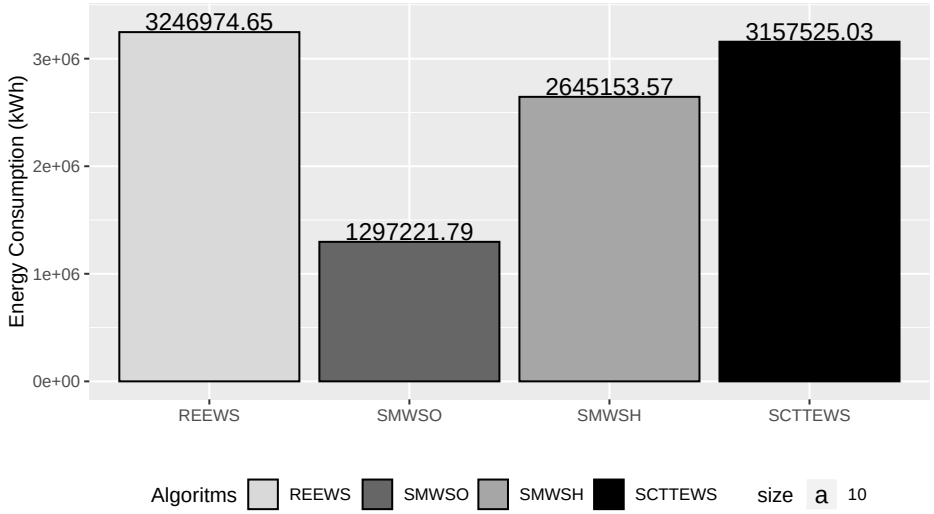
- *Homogeneity can produce better result if good instances are chosen for the execution of the workflow.* The performance the SMWSO algorithm compared to the one of the SMWSH algorithm confirm our statement. In fact, the two algorithms are designed almost in the same manner, apart from the variety of the types of resources used. While SMWSO uses an unique instance dynamically determined (called optimal instance type) among the available instance, SMWSH uses different instance determined as the scheduling process goes on. From Table 4.9 we see that both algorithm are sometime more energy-efficient that the other one, almost equitably. However, the different is not very significance when it is SMWSH that is more energy-efficient as it is in the case of SMWSO. The Figure 4.18c



(a) Total Energy consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for different workflow types.



(b) Total Energy consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for different workloads.



(c) Total Energy consumption of SCTTEWS, SMWSO and SMWSH vs REEWS, for all the experiments.

Figure 4.18: Total Energy consumption of SCTTEWS, SMWSO and SMWSH vs REEWS.

which presents the overall energy consumption of each of the algorithm demonstrates it, as well as the values of the different ANOVA and post hoc Tukey-Kramer tests on the energy efficiency (see the Tables 4.4, A.1, A.2, A.3, and A.4). Moreover, SMWSO significantly outperforms the three other algorithms with a total energy-saving (for all the experiments) of more than 50% (see Figure 4.18c).

- *If a suitable number of VMs is determined, it can help not only to produce better results in terms of success rate, but also upgrade the VM Utilization Maximization and the Energy Consumption Minimization as well as the Workload Maximization.* In our three multi-objective algorithms, the optimal number of VMs technique is proposed as an answer to that preoccupation and used to limit the number of VMs to use. The result show that our three algorithms significantly outperform REEWS in terms of energy-saving in most (80% of cases) of the types of workflow (apart one: Cybershake) and of the workload (see Table 4.9, and Figures 4.18a and 4.18b).

Furthermore, SMWSO is more stable no matter the workload of the workflow (see Figure 4.18b).

After SMWSO, SMWSH and SCTTEWS are the more energy-efficient compared to REEWS. From the ranking of Table 4.9 we see that SMWSH and SCTTEWS are equally more energy-efficient each one than the other according to the type and the workload of the workflow. However, SCTTEWS has a very poor energy-efficiency Epigenomics 1000 (see Figures 4.13a and 4.18b). As Epigenomics is far more energy consumer then the four other types of workflow, SMWSH scored a better energy-saving then SCTTEWS at overall (see Figure 4.18c).

Conclusion

In this chapter, we have proposed three multi-objective scheduling algorithms named Structure-based Multi-objective Workflow Scheduling (SMWSO), Structure-based Multi-objective Workflow Scheduling with heterogeneous instance types (SMWSH), and Structure-based Cost-Time trade-off and Energy-efficient Workflow Scheduling (SMWSH). The three algorithms aim at optimizing processing costs, makespan, and energy consumption, in the respect of the user-defined budget and deadline.

These algorithms rely on new concepts, the optimal number of VMs along with the optimal instance type helping to deal with the heterogeneous nature of cloud environments which is an opportunity and at the same time a real issue in cloud environments. These techniques aim at avoiding resource wastage by limiting the number of VMs to provision while giving relatively good performances in terms of makespan, execution cost, and energy consumption

minimization. The reduction of makespan contributes largely to the minimization of energy consumption. However, the parallel pipelines merging and the employment of the DVFS on non-critical paths have been used to further reduce the energy consumption.

While the SMWSO algorithm makes use of a unique VM instance type, which we called optimal instance type, the two others use several instance types for the workflow execution. However, the SCTTEWS algorithm employs the newly proposed technique, the Implicit Requested Instance Types Range (IRITR) evaluation to determine a range of VMs instance types that best suits the workflow execution. The IRITR evaluation helps to avoid overbidding as well as underbidding that may lead to budget and deadline violations respectively.

Since the workflow execution time is unknown before the end, estimation must be done in order to propose scheduling strategies. It is stated in the literature that there is no 100% accurate execution time estimation. Therefore, it is worth to mention that our makespan estimation still needs some investigation. However, it has been one major aspect of the optimal instance type evaluation.

Comparative experimentations have been done through simulations against the state-of-the-art algorithm REEWS, and the results presented and analysed based on ANOVA test along with pairwise tests using Tukey-Kramer. The results of the experiments prove the out-performance of our three proposals, SMWSO, SMWSH and SCTTEWS, in terms of energy-saving compared to the REEWS algorithm in 80% of cases. When we look at details, SMWSO, SMWSH and SCTTEWS scored almost equally the highest energy-saving for the different workflow and workload.

The results also proved that at overall, SMWSO is more than 50% the more energy-saving algorithm, followed by SMWSH, and then SCTTEWS. As for the success rate, even though SMWSO scored at overall the highest success rate, statistical tests proved that their is no significant difference between the four algorithms in terms of user satisfaction.

From all the above, we can say that the usage of the optimal number of VMs along with the optimal instance type, the IRITR evaluation, the task duplication, and the pipelines merging and slacking are relevant techniques for cost, makespan, and energy consumption reduction.

Resources homogeneity via the ”optimal instance type” technique helped producing better results. Also suitable number of VMs determined by the ”optimal number of VMs” helped producing better results. Those techniques have been designed using the structural properties of workflow and user requirements. Therefore, our hypotheses have been clearly validated by the outcome of our work.

Conclusion and Perspectives

Conclusion

Cloud computing has made the vision of IT resources as utility one step closer to reality. As a result, cloud data centers are expected to grow and accumulate a greater fraction of the world's IT resources. RightScale [110] in their 2015 report stated that there is a significant headroom for more enterprise workloads to move to the cloud. *68% of enterprises run less than a fifth of their application portfolios in the cloud while 55% of them has built a significant portion of their existing application portfolios with cloud-friendly architectures.* Just recently in 2020, Flexera [9] reported that public cloud adoption continues to accelerate, with nearly half of enterprise workloads and data in public clouds. In this context, the energy efficient management of data center resources is a crucial problem in terms of operating costs, return of investment (ROI) and CO_2 emissions. Inefficient and time-consuming processes remain one issue to handle in order to maximize the ROI of both cloud providers and cloud users [25].

This thesis is focused on the problem of scheduling the execution of scientific workflows on cloud data center resources. This problem is known to be NP-hard, specifically when we must effectively respect the multiple and contradictories constraints of users and cloud providers.

The main objective of this thesis was to propose scheduling strategies for the execution of scientific workflows in cloud computing data center in order to effectively reduce energy consumption, execution costs, and execution time. The set constraints of the scheduling were the respect of the user-defined budget and deadline. The expected effectiveness of the scheduling strategies should be independent of the type of workflow.

To this end, we have proposed consecutively five workflow scheduling algorithms relying on techniques based on the structural properties of workflows that we designed. The effectiveness of our proposals was analysed through comparatives simulations against state-of-the-art algorithms.

We firstly proposed a scheduling algorithm aiming at minimizing execution cost and execution time, with respect to the user-defined budget and deadline. The consideration was the execution of a single workflow on the resources of a cloud data center with a static VMs provisioning strategy. The algorithm, named Cost-Time Trade-off efficient Workflow Scheduling (CTTWS), consists of four main steps. The four steps are: task selection, Implicit Requested Instance Types Range (IRITR) evaluation, spare budget evaluation, and VM selection. The

IRITR evaluation which is a novel scheduling concept, aims at determining a range of VMs instance types that best suits the workflow execution, in order to avoid overbidding and underbidding that may lead to budget and deadline violation respectively. The IRITR evaluation relies on an analysis of the structure of the workflow and on the user-defined budget and deadline. Regarding the VM selection, we have used a fine grain trade-off function. Simulations results proved the effectiveness of our approach compared to a state-of-art algorithm, especially when there is a large variety of instance types. The results further highlight the importance of the IRITR. However, static provisioning though important for simplicity and a quick investigation is not suitable for cloud environments. Furthermore, we have realized that the algorithm has poor performance when it comes to a specific type of workflow.

Our second algorithm, the Cost-Time Trade-off efficient Workflow Scheduling with Dynamic provisioning (CTTWSDP) has been proposed to improve the CTTWS algorithm. CTTWSDP improves CTTWS in two main ways. Firstly, by proposing an improved Implicit Requested Instance Types Range (IRITR) evaluation. And secondly by using a dynamic VMs provisioning strategy with a limited number of leased VMs. The IRITR improvement as well as the limitation of the VMs in the dynamic provisioning rely on the determination of a limited number of VMs to use for the execution of the workflow. That number is the average of the number of tasks in the different levels of the workflow going from the root tasks to the exit tasks. The results of simulations prove the effectiveness of the proposal and highlight the improvement of CTTWSDP over CTTWS. Moreover, our algorithm achieves an average success rate higher from 17.09% to 44.80% when compared to three state-of-the-art algorithms. An analysis of the standard deviation of the success rate gives a standard deviation of 2.42 for CTTWSDP, which is smaller than the ones obtained by the other algorithms from 13.35 to 35.69. ANOVA along with post hoc Tukey-Kramer tests have proved that CTTWSDP is significantly more effective in terms of success rate (satisfaction of both user-defined budget and deadline).

Finally, we proposed three algorithms aiming at optimizing processing costs, makespan, and energy consumption, under user-defined budget and deadline constraints. The three algorithms are named: Structure-based Cost-Time trade-off and Energy efficient Workflow Scheduling (SCTTEWS), Structure-based Multi-objective Workflow Scheduling with an Optimal instance type (SMWSO), and Structure-based Multi-objective Workflow Scheduling with Heterogeneous instance types (SMWSH). The three algorithms rely on some structure-based techniques that we designed. Among these techniques we have the determination of an optimal number of VMs along with an optimal type of VM. Because of the precedence constraint between the tasks, it is not possible to have more tasks simultaneously in execution than the maximum number of tasks in a level. The optimal number of VMs is determined as a conditional function of the average, maximum, and the standard deviation of the number of tasks in the different levels of the workflow. That number is bounded by the average and the maximum number of tasks in the

levels. For the optimal type of VM, we determined the fastest instance type which can respect both the deadline and the budget of the user, according to a proposed makespan estimation. Other techniques are entry task duplication which helps in the reduction of the execution cost and time, and the parallel pipelines grouping and merging which uses the DVFS technique to reduce the energy consumption.

The SCTTEWS extends the CTTWSDP algorithm by handling the minimization of energy consumption in addition to the minimization of the execution cost and time. SCTTEWS improves the IRITR evaluation by using the optimal number of VMs instead of the average number of tasks in the levels like CTTWSDP. SCTTEWS also improve the trade-off function by adding weights to the two trade-off factors of the function. The weights are dynamically determined based on the structure of the workflow. In addition, SCTTEWS uses the entry task duplication and the parallel pipelines grouping and merging.

SMWSO makes use of one unique instance type for all the VMs created. It uses the determination of the optimal number of VMs and the determination of the optimal instance type. For the prioritization of the tasks, it uses a modified version of the up rank proposed by HEFT [82]. In addition, SMWSO uses the entry task duplication and the parallel pipelines grouping and merging.

SMWSH as for it behave almost like SMWSO, apart in the determination of the selection of the VM to uses for the execution of each task. Instead of using a homogeneous type for the VMs, makes use of different types of VMs. It employ a deadline distribution technique to share to user-define deadline among the tasks of the workflow. And for each task, according to its priority, determines the VM among the already created VMs, or the type of VMs which can end faster and in the respect of the sub-deadline of the task.

Comparative simulations have been made against the state-of-the-art algorithm Reliability and Energy Efficient Workflow scheduling (REEWS) using the five most spread workflow with five different workload for each. REEWS aim at minimizing energy consumption and maximizing the reliability of the system in the respect of the user-specified deadline.

Statistical tests using ANOVA along with Tukey-Kramer proved the significant out-performance of our three proposals in terms of energy-saving compared to REEWS, for 80% of workflow/workload scenarios. SMWSO, SMWSH and SCTTEWS scored almost equally the highest energy-saving for the different workflow and workload. However, SMWSO is at overall the more energy-saving algorithm with a total energy consumption that is more than 50% less than the one of the other, followed by SMWSH, and then SCTTEWS. In terms of user satisfaction (success rate), SMWSO scored at overall the highest performance, followed by REEWS. The SCTTEWS algorithm as for it realized good performance compared to SMWSH. Moreover, our three energy-efficient algorithms proved to be more effective than our bi-objective algorithm

CTTWSDP. However, statistical tests proved that there is no significant difference between the four algorithms SMWSO, SMWSH, SCTTEWS and REEWS in terms of success rate.

We have advocated that homogeneity can produce better result if good instances are chosen for the execution of the workflow. The performance the these algorithm confirm our statement. Also we had further advocated that if a suitable number of VMs is determined, it can help not only to produce better results, but also upgrade the VM Utilization Maximization and the Energy Consumption Minimization as well as the Workload Maximization. All these hypotheses have been clearly validated by the outcome of our work.

Perspectives for future works

Despite the substantial contributions of this thesis on minimizing energy consumption in the cloud as well as that of the executing cost and time of workflows, there are a number of open research challenges that must be addressed in order to improve the quality of our works.

Therefore in our future works, we will focus on the following:

- *Scheduling of multiple concurrent workflows*: In cloud environment, the workload is expected to arrive continuously, and workflows must be handled as soon as they arrive due to the QoS constraints defined by the users [68]. This implies dealing with the arrival of multiple workflows at real-time. Furthermore, the deadline constraint requirements and the dependencies between workflow tasks mean that unused time slots cannot be fully eliminated [41]. Some studies show that the execution of a single workflow on a set of processors leads to a wastage of resources because of the degree of parallelism [111]. As the provider might charge for all processors during the execution time, it is evidence that resources are being wasted because of the multiplicity of idle time slots when considering a single workflow [111]. The problem of scheduling multiple workflows in cloud environments is relatively recent [111, 111], and there is a need for more investigations.

- *Addressing the uncertainty nature of cloud environments using Machine Learning*: cloud environments are very dynamics due to performance variability. The virtualization, backbone of clouds is the primary source of the performance variability [34, 112]. Another source is the variation of the workload arriving towards cloud data centers. It is therefore very important to consider the uncertainty nature of cloud environments to have more realistic and mature solutions. For instance, the provisioning of a VM is subject to a creation delay. Mao and Humphrey studied VM startup time in 2012 by considering three cloud providers (Amazon EC2, Windows Azure, and Rackspace) [103] and reported that the average provisioning delay of a VM was 97. In 2016, Jones et al. [113] presented a

study that shows that three different cloud management frameworks (OpenStack, OpenNebula, and Eucalyptus) produced VM provisioning delays between 12 to 120 seconds. Network resources are also subjected to performance variation, with studies reporting a data transfer time variation of 19% in Amazon EC2 [114].

- *Using hybrid heuristic/Meta-heuristic approaches adapted dynamic and uncertainty nature of cloud*: Even though meta-heuristics approaches are more suitable for dealing with NP-hard problem, heuristics are preferable in cloud environment because of the dynamicity and uncertainty nature of cloud environments. However, it has been advocated to use hybrid solutions to take advantage of the strengths of both meta-heuristics and heuristics [3]. It has also been advised [31, 81] to work on the optimization of search time known to be long in meta-heuristic approaches, and design scalable approaches in order to deal with the challenges specific to cloud environments. We intend therefore to investigate on the design of hybrid solution to enhance the quality of our algorithms.

- *Reliability and Fault-tolerant management*: during workflow execution it is necessary to insure reliability and fault-tolerant. The uncertain nature of the cloud as well as material deficiency or computation imprecision may lead to failure. Therefore, it is essential to pay attention to reliability and fault-tolerant while managing cloud resources.

- *Internet of Things (IoT) / Edge computing Workflows*: One of the top cloud-edge computing opportunities is edge analytics, which allows data analysis and decision making closer to where the data is generated such as IoT devices, edge and gateways while the cloud is still responsible for the overall service life cycle management, service scheduling, data storage or warehousing, and more comprehensive service or data analysis such as big data analytics. The key benefits of edge analytics are reduced latency and increased security of the analytics, and hence, quick business decision making and so on [115]. While they are a good opportunity for business and scientific advancement, IoT/Cloud-edge computing obviously brings many technical challenges to cloud providers, especially in support of heterogeneous computing, network instability and security on the edge.

- *Adaptation of our algorithms for usage in production environments*: we have validated the performance of our algorithms through simulation, which is known as an acceptable approach. However, we plan to integrate and execute our proposed algorithms in existing workflow management systems such as the Cloudbus Toolkit [116], and cloud solutions like OpenStack[3].

[3]OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a data center, all managed and provisioned through APIs. https://www.openstack.org/

Bibliography

[1] R. Buyya, C. Vecchiola, S. T. Selvi, Mastering cloud computing: foundations and applications programming, Newnes, 2013.

[2] A. Beloglazov, R. Buyya, Y. C. Lee, A. Zomaya, A taxonomy and survey of energy-efficient data centers and cloud computing systems, in: Advances in computers, Vol. 82, Elsevier, 2011, pp. 47–111.

[3] M. A. Rodriguez, R. Buyya, A taxonomy and survey on scheduling algorithms for scientific workflows in iaas cloud computing environments, Concurrency and Computation: Practice and Experience 29 (8) (2017) e4041.

[4] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, K. Vahi, Characterizing and profiling scientific workflows, Future Generation Computer Systems 29 (3) (2013) 682–692.

[5] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, K. Vahi, Characterization of scientific workflows, in: 2008 third workshop on workflows in support of large-scale science, IEEE, 2008, pp. 1–10.

[6] Intel Corporation, Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor White Paper, `http://download.intel.com/design/network/papers/30117401.pdf`, online; accessed 29 May 2020 (2004).

[7] R. Buyya, Introduction to the ieee transactions on cloud computing, IEEE Transactions on cloud computing 1 (1) (2013) 3–21.

[8] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility, Future Generation computer systems 25 (6) (2009) 599–616.

[9] Flexera, State of the cloud report, `https://resources.flexera.com/web/pdf/report-state-of-the-cloud-2020.pdf`, online; accessed 28 June 2020 (2020).

[10] P. Delforge, America's Data Centers Consuming and Wasting Growing Amounts of Energy, `https://www.nrdc.org/resources/americas-data-centers-consuming-and-wasting-growing-amounts-energy`, online; accessed 19 December 2019 (2015).

[11] K. Djemame, D. Armstrong, R. Kavanagh, A. J. Ferrer, D. G. Perez, D. Antona, J.-C. Deprez, C. Ponsard, D. Ortiz, M. Macias, et al., Energy efficiency embedded service lifecycle: Towards an energy efficient cloud computing architecture, in: CEUR Workshop Proceedings, Vol. 1203, CEUR Workshop Proceedings, 2014, pp. 1–6.

[12] Greenpeace, How Clean is Your Cloud?, http://www.greenpeace.org/international/en/publications/Campaign-reports/Climate-Reports/How-Clean-is-Your-Cloud/, online; accessed 19 December 2019 (2012).

[13] T. V. T. Duy, Y. Sato, Y. Inoguchi, Performance evaluation of a green scheduling algorithm for energy savings in cloud computing, in: 2010 IEEE international symposium on parallel & distributed processing, workshops and Phd forum (IPDPSW), IEEE, 2010, pp. 1–8.

[14] A. Greenberg, J. Hamilton, D. A. Maltz, P. Patel, The cost of a cloud: research problems in data center networks (2008).

[15] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, M. A. Kozuch, Heterogeneity and dynamicity of clouds at scale: Google trace analysis, in: Proceedings of the Third ACM Symposium on Cloud Computing, 2012, pp. 1–13.

[16] L. A. Barroso, U. Hölzle, The case for energy-proportional computing, Computer 40 (12) (2007) 33–37.

[17] S. F. Piraghaj, A. V. Dastjerdi, R. N. Calheiros, R. Buyya, A survey and taxonomy of energy efficient resource management techniques in platform as a service cloud, in: Handbook of Research on End-to-End Cloud Computing Architecture Design, IGI Global, 2017, pp. 410–454.

[18] N. Garg, D. Singh, M. S. Goraya, Energy and resource efficient workflow scheduling in a virtualized cloud environment, Cluster Computing.

[19] M. Dabbagh, B. Hamdaoui, M. Guizani, A. Rayes, Toward energy-efficient cloud computing: Prediction, consolidation, and overcommitment., IEEE network 29 (2) (2015) 56–61.

[20] A. Beloglazov, J. Abawajy, R. Buyya, Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing, Future generation computer systems 28 (5) (2012) 755–768.

[21] R. Garg, M. Mittal, et al., Reliability and energy efficient workflow scheduling in cloud environment, Cluster Computing 22 (4) (2019) 1283–1297.

[22] Z. Li, J. Ge, H. Hu, W. Song, H. Hu, B. Luo, Cost and energy aware scheduling algorithm for scientific workflows with deadline constraint in clouds, IEEE Transactions on Services Computing 11 (4) (2015) 713–726.

[23] Z. Tang, Z. Cheng, K. Li, K. Li, An efficient energy scheduling algorithm for workflow tasks in hybrids and dvfs-enabled cloud environment, in: 2014 Sixth International Symposium on Parallel Architectures, Algorithms and Programming, IEEE, 2014, pp. 255–261.

[24] T. Guérout, T. Monteil, G. Da Costa, R. N. Calheiros, R. Buyya, M. Alexandru, Energy-aware simulation with dvfs, Simulation Modelling Practice and Theory 39 (2013) 76–91.

[25] Flexera, Maximizin cloud roi, `https://resources.flexera.com/web/pdf/WhitePaper-SLO-CM-Maximizing-Cloud-ROI-Hobson.pdf`, online; accessed 28 June 2020 (2019).

[26] M. Zakarya, L. Gillam, An energy aware cost recovery approach for virtual machine migration, in: International Conference on the Economics of Grids, Clouds, Systems, and Services, Springer, 2016, pp. 175–190.

[27] A. Beloglazov, Energy-efficient management of virtual machines in data centers for cloud computing, Ph.D. thesis (2013).

[28] M. Zakarya, An extended energy-aware cost recovery approach for virtual machine migration, IEEE Systems Journal 13 (2) (2018) 1466–1477.

[29] A. A. Khan, M. Zakarya, R. Buyya, R. Khan, M. Khan, O. Rana, An energy and performance aware consolidation technique for containerized datacenters, IEEE Transactions on Cloud Computing.

[30] G. Ismayilov, H. R. Topcuoglu, Neural network based multi-objective evolutionary algorithm for dynamic workflow scheduling in cloud computing, Future Generation Computer Systems 102 (2020) 307–322.

[31] M. H. Hilman, Budget-constrained workflow applications scheduling in workflow-as-a-service cloud computing environments, Ph.D. thesis (2020).

[32] E. Deelman, G. Singh, M. Livny, B. Berriman, J. Good, The cost of doing science on the cloud: the montage example, in: SC'08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing, IEEE, 2008, pp. 1–12.

[33] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. P. Berman, P. Maechling, Scientific workflow applications on amazon ec2, in: 2009 5th IEEE international conference on e-science workshops, IEEE, 2009, pp. 59–66.

[34] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, N. J. Wright, Performance analysis of high performance computing applications on the amazon web services cloud, in: 2nd IEEE international conference on cloud computing technology and science, IEEE, 2010, pp. 159–168.

[35] J.-S. Vöckler, G. Juve, E. Deelman, M. Rynge, B. Berriman, Experiences using cloud computing for a scientific workflow application, in: Proceedings of the 2nd international workshop on Scientific cloud computing, ACM, 2011, pp. 15–24.

[36] S. N. Srirama, P. Jakovits, E. Vainikko, Adapting scientific computing problems to clouds using mapreduce, Future Generation Computer Systems 28 (1) (2012) 184–192.

[37] R. Madduri, K. Chard, R. Chard, L. Lacinski, A. Rodriguez, D. Sulakhe, D. Kelly, U. Dave, I. Foster, The globus galaxies platform: delivering science gateways as a service, Concurrency and Computation: Practice and Experience 27 (16) (2015) 4344–4360.

[38] B. Balis, K. Figiela, K. Jopek, M. Malawski, M. Pawlik, Porting hpc applications to the cloud: A multi-frontal solver case study, Journal of Computational Science 18 (2017) 106–116.

[39] I. J. Taylor, E. Deelman, D. B. Gannon, M. Shields, et al., Workflows for e-Science: scientific workflows for grids, Vol. 1, Springer, 2007.

[40] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, J. Myers, Examining the challenges of scientific workflows, Computer 40 (12) (2007) 24–32.

[41] M. A. Rodriguez, R. Buyya, Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms, Future Generation Computer Systems 79 (2018) 739–750.

[42] H. Kimura, M. Sato, Y. Hotta, T. Boku, D. Takahashi, Emprical study on reducing energy of parallel programs using slack reclamation by dvfs in a power-scalable high performance cluster, in: 2006 IEEE international conference on cluster computing, IEEE, 2006, pp. 1–10.

[43] G. L. Stavrinides, H. D. Karatza, An energy-efficient, qos-aware and cost-effective scheduling approach for real-time workflow applications in cloud computing systems utilizing dvfs and approximate computations, Future Generation Computer Systems 96 (2019) 216–226.

[44] H. R. Faragardi, M. R. S. Sedghpour, S. Fazliahmadi, T. Fahringer, N. Rasouli, Grp-heft: A budget-constrained resource provisioning scheme for workflow scheduling in iaas clouds, IEEE Transactions on Parallel and Distributed Systems 31 (2019) 1239–1254.

[45] V. Singh, I. Gupta, P. K. Jana, A novel cost-efficient approach for deadline-constrained workflow scheduling by dynamic provisioning of resources, Future Generation Computer Systems 79 (2018) 95–110.

[46] I. Foster, Y. Zhao, I. Raicu, S. Lu, Cloud computing and grid computing 360-degree compared, in: Grid Computing Environments Workshop, 2008. GCE'08, Ieee, 2008, pp. 1–10.

[47] R. Buyya, J. Broberg, A. M. Goscinski, Cloud computing: Principles and paradigms, Vol. 87, John Wiley & Sons, 2010.

[48] A. Khosravi, S. K. Garg, R. Buyya, Energy and carbon-efficient placement of virtual machines in distributed cloud data centers, in: European Conference on Parallel Processing, Springer, 2013, pp. 317–328.

[49] R. Buyya, A. Beloglazov, J. Abawajy, Energy-efficient management of data center resources for cloud computing: a vision, architectural elements, and open challenges, arXiv preprint arXiv:1006.0308.

[50] L. M. Vaquero, L. Rodero-Merino, J. Caceres, M. Lindner, A break in the clouds: towards a cloud definition, ACM SIGCOMM Computer Communication Review 39 (1) (2008) 50–55.

[51] P. Mell, T. Grance, et al., The nist definition of cloud computing.

[52] K. Chandrasekaran, Essentials of cloud computing, Chapman and Hall/CRC, 2014.

[53] T. Guérout, Ordonnancement sous contraintes de qualité de service dans les clouds, Ph.D. thesis (2014).

[54] P. T. Endo, G. E. Gonçalves, D. Rosendo, D. Gomes, G. L. Santos, A. L. C. Moreira, J. Kelner, D. Sadok, M. Mahloo, Highly available clouds: System modeling, evaluations, and open challenges, in: Research Advances in Cloud Computing, Springer, 2017, pp. 21–53.

[55] Y. Zhang, K. Chakrabarty, Energy-aware adaptive checkpointing in embedded real-time systems, in: 2003 Design, Automation and Test in Europe Conference, 2003, pp. 918–923.

[56] G. E. Moore, Cramming more components onto integrated circuits, Proceedings of the IEEE 86 (1) (1998) 82–85.

[57] A. Beloglazov, R. Buyya, Energy efficient allocation of virtual machines in cloud data centers, in: 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, IEEE, 2010, pp. 577–578.

[58] GigeNET, Cloud Computing Is Still Dangerously Underutilized, `https://www.gigenet.com/blog/underutilizing-cloud-computing-resources`, online; accessed 19 December 2019 (2019).

[59] S. Herbert, D. Marculescu, Analysis of dynamic voltage/frequency scaling in chip-multiprocessors, in: Proceedings of the 2007 international symposium on Low power electronics and design (ISLPED'07), IEEE, 2007, pp. 38–43.

[60] Y. C. Lee, A. Y. Zomaya, Energy conscious scheduling for distributed computing systems under different operating conditions, IEEE Transactions on Parallel and Distributed Systems 22 (8) (2011) 1374–1381.

[61] T. Pering, T. Burd, R. Brodersen, The simulation and evaluation of dynamic voltage scaling algorithms, in: Proceedings. 1998 International Symposium on Low Power Electronics and Design (IEEE Cat. No. 98TH8379), IEEE, 1998, pp. 76–81.

[62] P. E. Allen, D. R. Holberg, Cmos analog circuit design, oxford university press (2002).

[63] S. Bansal, P. Kumar, K. Singh, An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems, IEEE Transactions on Parallel and Distributed Systems 14 (6) (2003) 533–544.

[64] T. Ishihara, H. Yasuura, Voltage scheduling problem for dynamically variable voltage processors, in: Proceedings of the 1998 international symposium on Low power electronics and design, 1998, pp. 197–202.

[65] T. D. Burd, R. W. Brodersen, Design issues for dynamic voltage scaling, in: Proceedings of the 2000 international symposium on Low power electronics and design, 2000, pp. 9–14.

[66] M. Adhikari, T. Amgoth, S. N. Srirama, A survey on scheduling strategies for workflows in cloud environment and emerging trends, ACM Computing Surveys (CSUR) 52 (4) (2019) 1–36.

[67] R. F. da Silva, R. Filgueira, I. Pietri, M. Jiang, R. Sakellariou, E. Deelman, A characterization of workflow management systems for extreme-scale applications, Future Generation Computer Systems 75 (2017) 228–238.

[68] M. H. Hilman, M. A. Rodriguez, R. Buyya, Multiple workflows scheduling in multi-tenant distributed systems: A taxonomy and future directions, ACM Computing Surveys (CSUR) 53 (1) (2020) 1–39.

[69] J. Yu, R. Buyya, A taxonomy of workflow management systems for grid computing, Journal of Grid Computing 3 (3-4) (2005) 171–200.

[70] J. D. Ullman, Np-complete scheduling problems, Journal of Computer and System sciences 10 (3) (1975) 384–393.

[71] A. Abramovici, W. E. Althouse, R. W. Drever, Y. Gürsel, S. Kawamura, F. J. Raab, D. Shoemaker, L. Sievers, R. E. Spero, K. S. Thorne, et al., Ligo: The laser interferometer gravitational-wave observatory, science 256 (5055) (1992) 325–333.

[72] J. Livny, H. Teonadi, M. Livny, M. K. Waldor, High-throughput, kingdom-wide prediction and annotation of bacterial non-coding rnas, PloS one 3 (9) (2008) e3197.

[73] A. N. Njoya, W. Abdou, A. Dipanda, E. Tonye, Optimization of sensor deployment using multi-objective evolutionary algorithms, Journal of Reliable Intelligent Environments 2 (4) (2016) 209–220.

[74] W.-N. Chen, J. Zhang, An ant colony optimization approach to a grid workflow scheduling problem with various qos requirements, IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 39 (1) (2008) 29–43.

[75] X.-H. Hu, J.-C. Ouyang, Z.-H. Yang, Z.-H. Chen, An ipso algorithm for grid task scheduling based on satisfaction rate, in: 2009 International Conference on Intelligent Human-Machine Systems and Cybernetics, Vol. 1, IEEE, 2009, pp. 262–265.

[76] L.-Y. Chuang, S.-W. Tsai, C.-H. Yang, Catfish particle swarm optimization, in: 2008 IEEE Swarm Intelligence Symposium, IEEE, 2008, pp. 1–5. doi:http://dx.doi.org/10.1109/SIS.2008.4668277.

[77] S. Pandey, L. Wu, S. M. Guru, R. Buyya, A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments, in: 2010 24th IEEE international conference on advanced information networking and applications, IEEE, 2010, pp. 400–407.

[78] M. A. Rodriguez, R. Buyya, Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds, IEEE transactions on cloud computing 2 (2) (2014) 222–235.

[79] S. Elsherbiny, E. Eldaydamony, M. Alrahmawy, A. E. Reyad, An extended intelligent water drops algorithm for workflow scheduling in cloud computing environment, Egyptian informatics journal 19 (1) (2018) 33–55.

[80] A. Verma, S. Kaushal, A hybrid multi-objective particle swarm optimization for scientific workflow scheduling, Parallel Computing 62 (2017) 1–19.

[81] V. Arabnejad, K. Bubendorfer, B. Ng, Budget and deadline aware e-science workflow scheduling in clouds, IEEE Transactions on Parallel and Distributed Systems 30 (1) (2019) 29–44.

[82] H. Topcuoglu, S. Hariri, M.-y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE transactions on parallel and distributed systems 13 (3) (2002) 260–274.

[83] D. Poola, S. K. Garg, R. Buyya, Y. Yang, K. Ramamohanarao, Robust scheduling of scientific workflows with deadline and budget constraints in clouds, in: 2014 IEEE 28th international conference on advanced information networking and applications, IEEE, 2014, pp. 858–865.

[84] W. Zheng, R. Sakellariou, Budget-deadline constrained workflow planning for admission control, Journal of grid computing 11 (4) (2013) 633–651.

[85] A. Verma, S. Kaushal, Cost-time efficient scheduling plan for executing workflows in the cloud, Journal of grid computing 13 (4) (2015) 495–506.

[86] F. Abazari, M. Analoui, H. Takabi, S. Fu, Mows: Multi-objective workflow scheduling in cloud computing based on heuristic algorithm, Simulation Modelling Practice and Theory 93 (2019) 119–132.

[87] V. Arabnejad, K. Bubendorfer, B. Ng, Scheduling deadline constrained scientific workflows on dynamically provisioned cloud resources, Future Generation Computer Systems 75 (2017) 348–364.

[88] M. A. Khan, Scheduling for heterogeneous systems using constrained critical paths, Parallel Computing 38 (4-5) (2012) 175–193.

[89] Z. Zhu, X. Tang, Deadline-constrained workflow scheduling in iaas clouds with multi-resource packing, Future Generation Computer Systems 101 (2019) 880–893.

[90] Y. Pan, S. Wang, L. Wu, Y. Xia, W. Zheng, S. Pang, Z. Zeng, P. Chen, Y. Li, A novel approach to scheduling workflows upon cloud resources with fluctuating performance, Mobile Networks and Applications (2020) 1–11.

[91] Q. Huang, S. Su, J. Li, P. Xu, K. Shuang, X. Huang, Enhanced energy-efficient scheduling for parallel applications in cloud, in: 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), IEEE, 2012, pp. 781–786.

[92] J. J. Durillo, H. M. Fard, R. Prodan, Moheft: A multi-objective list-based method for workflow scheduling, in: 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings, IEEE, 2012, pp. 185–192.

[93] V. Singh, I. Gupta, P. K. Jana, An energy efficient algorithm for workflow scheduling in iaas cloud, Journal of Grid Computing (2019) 1–20.

[94] R. Chard, K. Chard, K. Bubendorfer, L. Lacinski, R. Madduri, I. Foster, Cost-aware cloud provisioning, in: 2015 IEEE 11th International Conference on e-Science, IEEE, 2015, pp. 136–144.

[95] Amazon EC2, Amazon EC2 Instance Types, `https://aws.amazon.com/ec2/instance-types/`, online; accessed 06 July 2019 (2019).

[96] Amazon EC2, Amazon EC2 pricing, `https://aws.amazon.com/ec2/pricing/on-demand/`, online; accessed 06 July 2019 (2019).

[97] Z. Li, J. Ge, H. Hu, W. Song, H. Hu, B. Luo, Cost and energy aware scheduling algorithm for scientific workflows with deadline constraint in clouds, IEEE Transactions on Services Computing 11 (4) (2018) 713–726.

[98] W. Chen, E. Deelman, Workflowsim: A toolkit for simulating scientific workflows in distributed environments, in: 2012 IEEE 8th International Conference on E-Science, IEEE, 2012, pp. 1–8.

[99] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, R. Buyya, Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Software: Practice and experience 41 (1) (2011) 23–50.

[100] C. B. Warner, A. M. Meehan, Microsoft excel™ as a tool for teaching basic statistics, Teaching of Psychology 28 (4) (2001) 295–298.

[101] M. Ghobaei-Arani, A. A. Rahmanian, M. S. Aslanpour, S. E. Dashti, Csa-wsc: cuckoo search algorithm for web service composition in cloud environments, Soft Computing 22 (24) (2018) 8353–8378.

[102] M. Ghobaei-Arani, A. A. Rahmanian, A. Souri, A. M. Rahmani, A moth-flame optimization algorithm for web service composition in cloud computing: simulation and verification, Software: Practice and Experience 48 (10) (2018) 1865–1892.

[103] M. Mao, M. Humphrey, A performance study on the vm startup time in the cloud, in: 2012 IEEE Fifth International Conference on Cloud Computing, IEEE, 2012, pp. 423–430.

[104] A. Choudhary, I. Gupta, V. Singh, P. K. Jana, A gsa based hybrid algorithm for bi-objective workflow scheduling in cloud computing, Future Generation Computer Systems 83 (2018) 14–26.

[105] Statology, How to Perform a Tukey-Kramer Post Hoc Test in Excel, `https://www.statology.org/tukey-kramer-post-hoc-test-excel/`, online; accessed 9 January 2021 (2020).

[106] G. L. Stavrinides, H. D. Karatza, Scheduling data-intensive workloads in large-scale distributed systems: trends and challenges, in: Modeling and Simulation in HPC and Cloud Systems, Springer, 2018, pp. 19–43.

[107] G. L. Stavrinides, H. D. Karatza, The impact of workload variability on the energy efficiency of large-scale heterogeneous distributed systems, Simulation Modelling Practice and Theory 89 (2018) 135–143.

[108] G. Wang, Y. Wang, H. Liu, H. Guo, Hsip: A novel task scheduling algorithm for heterogeneous computing, Scientific Programming 2016.

[109] M. Mao, M. Humphrey, Auto-scaling to minimize cost and meet application deadlines in cloud workflows, in: SC'11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2011, pp. 1–12.

[110] R. RightScale, State of the cloud report, Santa Barbara, California (https://assets.rightscale.com/uploads/pdfs/RightScale-2015-State-of-the-Cloud-Report.pdf).

[111] H. Arabnejad, J. G. Barbosa, Multi-qos constrained and profit-aware scheduling approach for concurrent workflows on heterogeneous systems, Future Generation Computer Systems 68 (2017) 211–221.

[112] P. Leitner, J. Cito, Patterns in the chaos—a study of performance variation and predictability in public iaas clouds, ACM Transactions on Internet Technology (TOIT) 16 (3) (2016) 1–23.

[113] M. Jones, B. Arcand, B. Bergeron, D. Bestor, C. Byun, L. Milechin, V. Gadepally, M. Hubbell, J. Kepner, P. Michaleas, et al., Scalability of vm provisioning systems, in: 2016 IEEE High Performance Extreme Computing Conference (HPEC), IEEE, 2016, pp. 1–5.

[114] J. Schad, J. Dittrich, J.-A. Quiané-Ruiz, Runtime measurements in the cloud: observing, analyzing, and reducing variance, Proceedings of the VLDB Endowment 3 (1-2) (2010) 460–471.

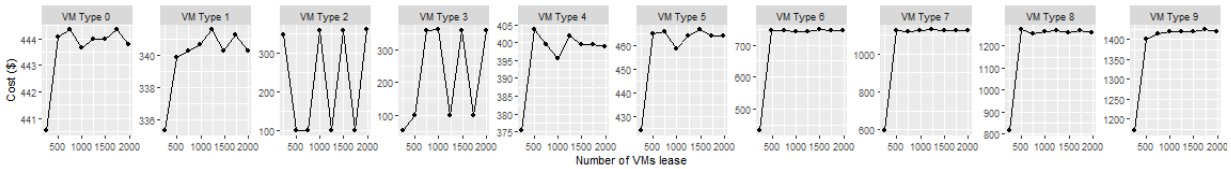
[115] Devops, Extending cloud to enable intelligent edge, `https://devops.com/extending-cloud-to-enable-intelligent-edge/`, online; accessed 04 June 2021 (2020).

[116] R. Buyya, S. Pandey, C. Vecchiola, Cloudbus toolkit for market-oriented cloud computing, in: IEEE International Conference on Cloud Computing, Springer, 2009, pp. 24–44.

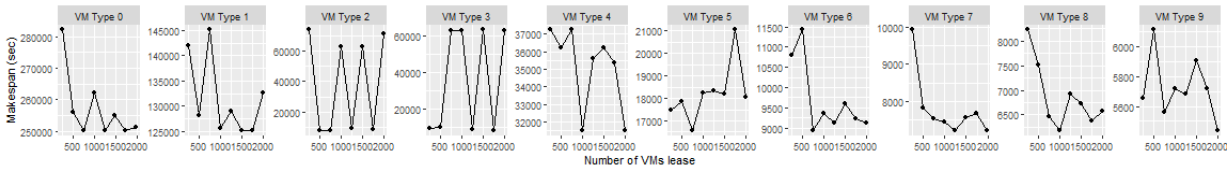
CHAPTER

A | Appendix A

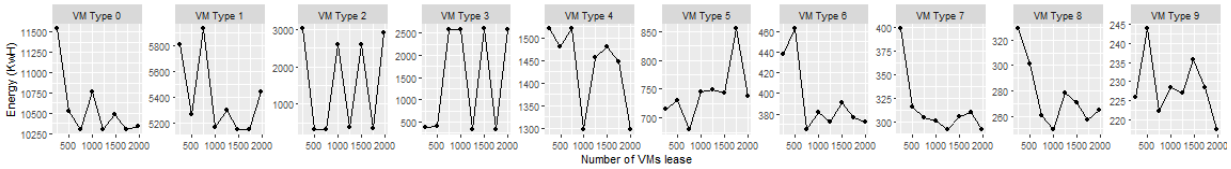
A.1 Analysis of the optimal number of VMs over execution cost, execution time, and energy consumption, when executing a workflow



(a) Cost per number of VMs leased



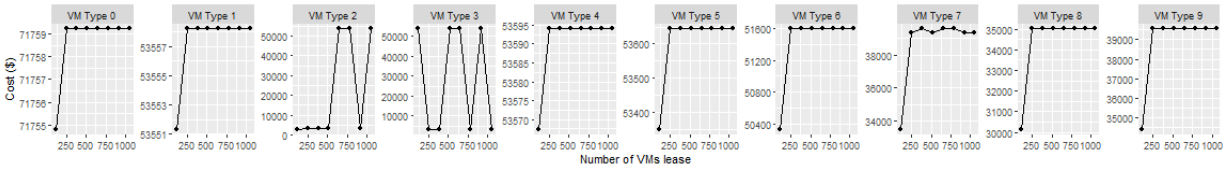
(b) Makespan per number of VMs leased



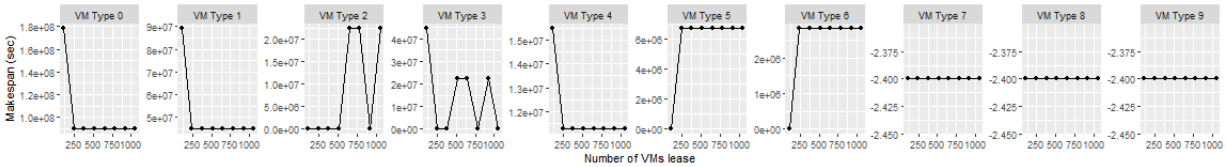
(c) Energy per number of VMs leased

Figure A.1: Cost, Makespan and Energy per number of VMs leased for CyberShake_1000.

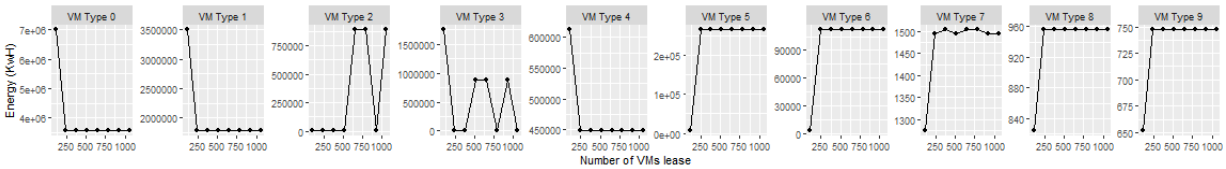
A.2 ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for CYBERSHAKE, EPIGENOMICS, SIPHT, and LIGO



(a) Cost per number of VMs leased

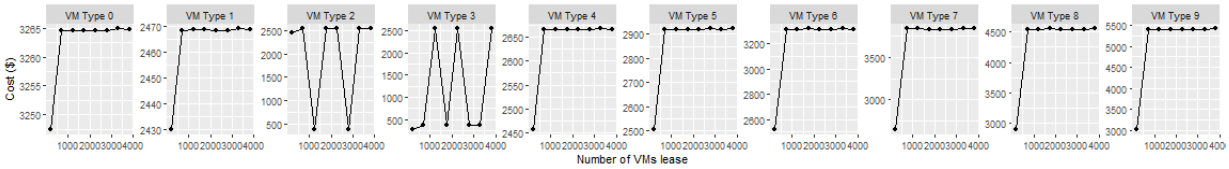


(b) Makespan per number of VMs leased

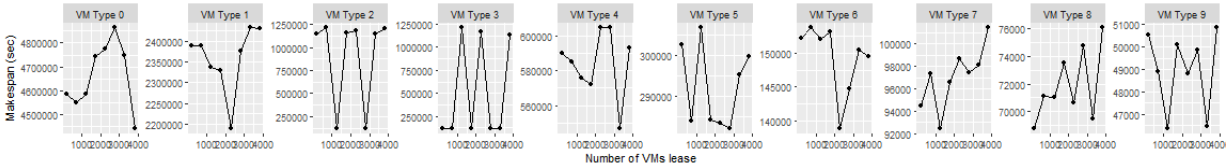


(c) Energy per number of VMs leased

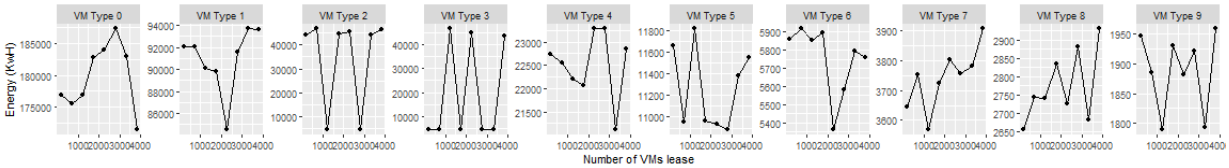
Figure A.2: Cost, Makespan and Energy per number of VMs leased for Epigenomics_1000.



(a) Cost per number of VMs leased

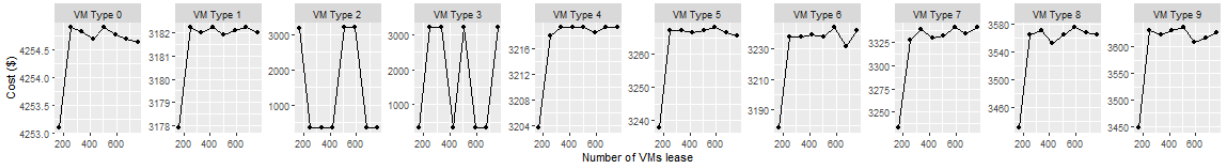


(b) Makespan per number of VMs leased

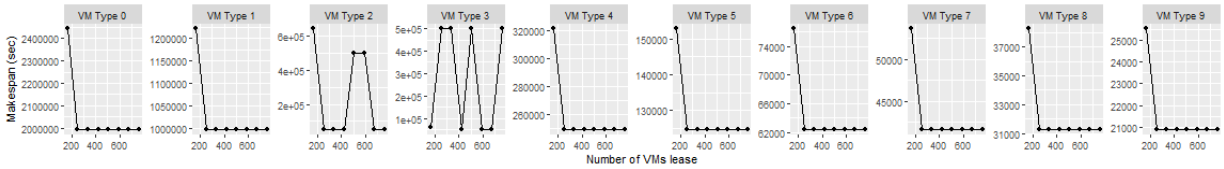


(c) Energy per number of VMs leased

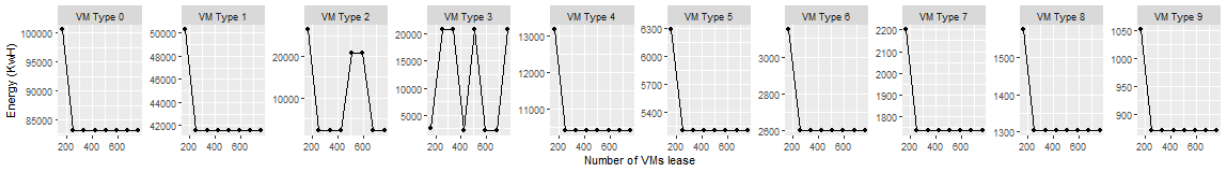
Figure A.3: Cost, Makespan and Energy per number of VMs leased for Sipht_1000.



(a) Cost per number of VMs leased



(b) Makespan per number of VMs leased



(c) Energy per number of VMs leased

Figure A.4: Cost, Makespan and Energy per number of VMs leased for Inspiral_1000.

Table A.1: ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for CYBERSHAKE 50, 100, 200, 500 and 10000

ANOVA input summary				
Group	***Count***	***Sum***	***Average***	***Variance***
SMWSO	16	317.05	19.81	11.69
SCTTEWS	16	262.11	16.38	33.88
SMWSH	16	201.76	12.61	1.35E-29
REEWS	16	183	11.44	0.71

ANOVA test result					
Source of Variation	***SS***	***df***	***MS***	***F stat***	***P-value***	***F crit***
Between Groups	695.81	3	231.94	20.04	4.05E-09	2.76
Within Groups	694.27	60	11.57			
Total	1390.08	63				

Tukey-Kramer pairwise between the four algorithms				
Comparison	***Diff***	***Abs. Diff***	***Q crit***	***Is Sig?***
SMWSO vs SCTTEWS	3.43	3.43	3.18	**YES**
SMWSO vs SMWSH	7.20	7.20	3.18	**YES**
SMWSO vs REEWS	8.38	8.38	3.18	**YES**
SCTTEWS vs SMWSH	3.77	3.77	3.18	**YES**
SCTTEWS vs REEWS	4.94	4.94	3.18	**YES**
SMWSH vs REEWS	1.17	1.17	3.18	NO

Algorithm Ranking in terms of Energy saving : SMWSO $\rightarrowtail 4$ | ***SCTTEWS*** $\rightarrowtail 3$ | ***SMWSH*** $\rightarrowtail 1$ | ***REEWS*** $\rightarrowtail 1$

(a) ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for **CYBERSHAKE 50**

ANOVA input summary				
Group	***Count***	***Sum***	***Average***	***Variance***
SMWSO	16	824.99	51.56	80.97
SCTTEWS	16	452.59	28.29	2.29E-05
SMWSH	16	452.59	28.29	2.29E-05
REEWS	16	421.84	26.36	3.95

ANOVA test result					
Source of Variation	***SS***	***df***	***MS***	***F stat***	***P-value***	***F crit***
Between Groups	6902.88	3	2300.96	108.37	3.44E-24	2.76
Within Groups	1273.88	60	21.23			
Total	8176.76	63				

Tukey-Kramer pairwise between the four algorithms				
Comparison	***Diff***	***Abs. Diff***	***Q crit***	***Is Sig?***
SMWSO vs SCTTEWS	23.27	23.27	4.30	**YES**
SMWSO vs SMWSH	23.27	23.27	4.30	**YES**
SMWSO vs REEWS	25.20	25.20	4.30	**YES**
SCTTEWS vs SMWSH	0	0	4.30	NO
SCTTEWS vs REEWS	1.92	1.92	4.30	NO
SMWSH vs REEWS	1.92	1.92	4.30	NO

Algorithm Ranking in terms of Energy saving : SMWSO $\rightarrowtail 4$ | ***SCTTEWS*** $\rightarrowtail 1$ | ***SMWSH*** $\rightarrowtail 1$ | ***REEWS*** $\rightarrowtail 1$

(b) ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for **CYBERSHAKE 100**

ANOVA input summary				
Group	***Count***	***Sum***	***Average***	***Variance***
SMWSO	16	1228.98	76.81125	544.51
SCTTEWS	16	838.65	52.415625	0.04
SMWSH	16	840.08	52.505	9.63E-3
REEWS	16	788.5	49.28125	43.56

ANOVA test result					
Source of Variation	***SS***	***df***	***MS***	***F stat***	***P-value***	***F crit***
Between Groups	7856.26	3	2618.75	17.81	2.20E-08	2.76
Within Groups	8821.89	60	147.03			
Total	16678.15	63				

Tukey-Kramer pairwise between the four algorithms				
Comparison	***Diff***	***Abs. Diff***	***Q crit***	***Is Sig?***
SMWSO vs SCTTEWS	24.39	24.39	11.33	**YES**
SMWSO vs SMWSH	24.31	24.31	11.33	**YES**
SMWSO vs REEWS	27.53	27.53	11.33	**YES**
SCTTEWS vs SMWSH	-0.09	0.09	11.33	NO
SCTTEWS vs REEWS	3.13	3.13	11.33	NO
SMWSH vs REEWS	3.22	3.22	11.33	NO

Algorithm Ranking in terms of Energy saving : SMWSO $\rightarrowtail 4$ | ***SCTTEWS*** $\rightarrowtail 1$ | ***SMWSH*** $\rightarrowtail 1$ | ***REEWS*** $\rightarrowtail 1$

(c) ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for **CYBERSHAKE 200**

ANOVA input summary				
Group	***Count***	***Sum***	***Average***	***Variance***
SMWSO	16	4358.45	272.40	6975.37
SCTTEWS	16	2544.83	159.05	68.87
SMWSH	16	2625.71	164.11	11.06
REEWS	16	2169.12	135.57	438.78

ANOVA test result					
Source of Variation	***SS***	***df***	***MS***	***F stat***	***P-value***	***F crit***
Between Groups	178764.77	3	59588.26	31.80	1.97E-12	2.76
Within Groups	112411.28	60	1873.52			
Total	291176.04	63				

Tukey-Kramer pairwise between the four algorithms				
Comparison	***Diff***	***Abs. Diff***	***Q crit***	***Is Sig?***
SMWSO vs SCTTEWS	113.35	113.35	40.44	**YES**
SMWSO vs SMWSH	108.30	108.30	40.44	**YES**
SMWSO vs REEWS	136.83	136.83	40.44	**YES**
SCTTEWS vs SMWSH	-5.05	5.05	40.44	NO
SCTTEWS vs REEWS	23.48	23.48	40.44	NO
SMWSH vs REEWS	28.56	28.56	40.44	NO

Algorithm Ranking in terms of Energy saving : SMWSO $\rightarrowtail 4$ | ***SCTTEWS*** $\rightarrowtail 1$ | ***SMWSH*** $\rightarrowtail 1$ | ***REEWS*** $\rightarrowtail 1$

(d) ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for **CYBERSHAKE 500**

ANOVA input summary				
Group	***Count***	***Sum***	***Average***	***Variance***
SMWSO	16	9447.19	590.45	32801.12
SCTTEWS	16	6600.18	412.51	239.27
SMWSH	16	7097.64	443.60	151.72
REEWS	16	4640.66	290.04	1933.73

ANOVA test result					
Source of Variation	***SS***	***df***	***MS***	***F stat***	***P-value***	***F crit***
Between Groups	732070.59	3	244023.53	27.79	2.16E-11	2.76
Within Groups	526887.62	60	8781.46			
Total	1258958.21	63				

Tukey-Kramer pairwise between the four algorithms				
Comparison	***Diff***	***Abs. Diff***	***Q crit***	***Is Sig?***
SMWSO vs SCTTEWS	177.94	177.94	87.55	**YES**
SMWSO vs SMWSH	146.85	146.85	87.55	**YES**
SMWSO vs REEWS	300.41	300.41	87.55	**YES**
SCTTEWS vs SMWSH	-31.09	31.09	87.55	NO
SCTTEWS vs REEWS	122.47	122.47	87.55	**YES**
SMWSH vs REEWS	153.56	153.56	87.55	**YES**

Algorithm Ranking in terms of Energy saving : SMWSO $\rightarrowtail 4$ | ***SCTTEWS*** $\rightarrowtail 2$ | ***SMWSH*** $\rightarrowtail 2$ | ***REEWS*** $\rightarrowtail 1$

(e) ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for **CYBERSHAKE 1000**

Table A.2: ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for EPIGENOMICS 50, 100, 200, 500 and 10000

ANOVA input summary				
Group	***Count***	***Sum***	***Average***	***Variance***
SMWSO	16	7737.44	483.59	1.38E-26
SCTTEWS	16	11619.79	726.24	44156.36
SMWSH	16	5983.04	373.94	1.38E-26
REEWS	16	16719.67	1044.98	39364.55

ANOVA test result						
Source of Variation	***SS***	***df***	***MS***	***F stat***	***P-value***	***F crit***
Between Groups	4248249.48	3	1416083.16	67.82	2.92E-19	2.76
Within Groups	1252813.72	60	20880.23			
Total	5501063.204	63				

Tukey-Kramer pairwise between the four algorithms				
Comparison	***Diff***	***Abs. Diff***	***Q crit***	***Is Sig?***
SMWSO vs SCTTEWS	-242.65	242.65	135	**YES**
SMWSO vs SMWSH	109.65	109.65	135	NO
SMWSO vs REEWS	-561.39	561.39	135	**YES**
SCTTEWS vs SMWSH	352.30	352.30	135	**YES**
SCTTEWS vs REEWS	-318.74	318.74	135	**YES**
SMWSH vs REEWS	-671.04	671.04	135	**YES**

Algorithm Ranking in terms of Energy saving : SMWSO $\rightarrowtail 1$ | ***SCTTEWS*** $\rightarrowtail 4$ | ***SMWSH*** $\rightarrowtail 3$ | ***REEWS*** $\rightarrowtail 4$

(a) ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for **EPIGENOMICS 50**

ANOVA input summary				
Group	***Count***	***Sum***	***Average***	***Variance***
SMWSO	16	31319.04	1957.44	2.20E-25
SCTTEWS	16	31521.44	1970.09	0
SMWSH	16	31521.44	1970.09	0
REEWS	16	72331.6	4520.725	1273511.80

ANOVA test result						
Source of Variation	***SS***	***df***	***MS***	***F stat***	***P-value***	***F crit***
Between Groups	78328911.37	3	26109637.12	82.01	3.33E-21	2.76
Within Groups	19102676.96	60	318377.95			
Total	97431588.33	63				

Tukey-Kramer pairwise between the four algorithms				
Comparison	***Diff***	***Abs. Diff***	***Q crit***	***Is Sig?***
SMWSO vs SCTTEWS	23.27	23.27	527.16	NO
SMWSO vs SMWSH	23.27	23.27	527.16	NO
SMWSO vs REEWS	-2563.28	2563.28	527.16	**YES**
SCTTEWS vs SMWSH	0	0	527.16	NO
SCTTEWS vs REEWS	-2550.63	2550.63	527.16	**YES**
SMWSH vs REEWS	-2550.63	2550.63	527.16	**YES**

Algorithm Ranking in terms of Energy saving : SMWSO $\rightarrowtail 1$ | ***SCTTEWS*** $\rightarrowtail 1$ | ***SMWSH*** $\rightarrowtail 1$ | ***REEWS*** $\rightarrowtail 4$

(b) ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for **EPIGENOMICS 100**

ANOVA input summary				
Group	***Count***	***Sum***	***Average***	***Variance***
SMWSO	16	78973.92	4935.87	0
SCTTEWS	16	74173.28	4635.83	8.82E-25
SMWSH	16	74173.28	4635.83	8.82E-25
REEWS	16	157225.76	9826.61	3929058.29

ANOVA test result						
Source of Variation	***SS***	***df***	***MS***	***F stat***	***P-value***	***F crit***
Between Groups	311951119.1	3	103983706.4	105.86	6.23E-24	2.76
Within Groups	58935874.28	60	982264.57			
Total	370886993.4	63				

Tukey-Kramer pairwise between the four algorithms				
Comparison	***Diff***	***Abs. Diff***	***Q crit***	***Is Sig?***
SMWSO vs SCTTEWS	300.04	300.04	925.95	NO
SMWSO vs SMWSH	300.04	300.04	925.95	NO
SMWSO vs REEWS	27.53	27.53	925.95	**YES**
SCTTEWS vs SMWSH	0	0	925.95	NO
SCTTEWS vs REEWS	-5190.78	5190.78	925.95	**YES**
SMWSH vs REEWS	-5190.78	5190.78	925.95	**YES**

Algorithm Ranking in terms of Energy saving : SMWSO $\rightarrowtail 1$ | ***SCTTEWS*** $\rightarrowtail 1$ | ***SMWSH*** $\rightarrowtail 1$ | ***REEWS*** $\rightarrowtail 4$

(c) ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for **EPIGENOMICS 200**

ANOVA input summary				
Group	***Count***	***Sum***	***Average***	***Variance***
SMWSO	16	293251.68	18328.23	1.41E-23
SCTTEWS	16	170141.92	10633.87	3.53E-24
SMWSH	16	180450.4	11278.15	1.41E-23
REEWS	16	1040176.84	65011.05	781741689.1

ANOVA test result						
Source of Variation	***SS***	***df***	***MS***	***F stat***	***P-value***	***F crit***
Between Groups	32530842419	3	10843614140	55.483	2.67E-17	2.76
Within Groups	11726125336	60	195435422.3			
Total	44256967756	63				

Tukey-Kramer pairwise between the four algorithms				
Comparison	***Diff***	***Abs. Diff***	***Q crit***	***Is Sig?***
SMWSO vs SCTTEWS	7694.36	7694.36	13061	NO
SMWSO vs SMWSH	7050.08	7050.08	13061	NO
SMWSO vs REEWS	-46682.82	46682.82	13061	**YES**
SCTTEWS vs SMWSH	-644.28	644.28	13061	NO
SCTTEWS vs REEWS	-54377.18	54377.18	13061	**YES**
SMWSH vs REEWS	-53732.90	53732.90	13061	**YES**

Algorithm Ranking in terms of Energy saving : SMWSO $\rightarrowtail 1$ | ***SCTTEWS*** $\rightarrowtail 1$ | ***SMWSH*** $\rightarrowtail 1$ | ***REEWS*** $\rightarrowtail 4$

(d) ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for **EPIGENOMICS 500**

ANOVA input summary				
Group	***Count***	***Sum***	***Average***	***Variance***
SMWSO	16	726268	45391.75	0
SCTTEWS	16	2702695.68	168918.48	0
SMWSH	16	2188427.2	136776.7	9.03E-22
REEWS	16	1456497.32	91031.08	554564667.4

ANOVA test result						
Source of Variation	***SS***	***df***	***MS***	***F stat***	***P-value***	***F crit***
Between Groups	1.39E+11	3	46513617351	335.50	1.95E-37	2.76
Within Groups	8318470011	60	138641166.8			
Total	1.48E+11	63				

Tukey-Kramer pairwise between the four algorithms				
Comparison	***Diff***	***Abs. Diff***	***Q crit***	***Is Sig?***
SMWSO vs SCTTEWS	-123526.73	123526.73	11000.71	**YES**
SMWSO vs SMWSH	-91384.95	91384.95	11000.71	**YES**
SMWSO vs REEWS	-45639.33	45639.33	11000.71	**YES**
SCTTEWS vs SMWSH	32141.78	32141.78	11000.71	**YES**
SCTTEWS vs REEWS	77887.40	77887.40	11000.71	**YES**
SMWSH vs REEWS	45745.61	45745.61	11000.71	**YES**

Algorithm Ranking in terms of Energy saving : SMWSO $\rightarrowtail 1$ | ***SCTTEWS*** $\rightarrowtail 4$ | ***SMWSH*** $\rightarrowtail 3$ | ***REEWS*** $\rightarrowtail 2$

(e) ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for **EPIGENOMICS 1000**

Table A.3: ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for SIPHT 50, 100, 200, 500 and 10000

ANOVA input summary				
Group	***Count***	***Sum***	***Average***	***Variance***
SMWSO	16	1694.08	105.88	0.17
SCTTEWS	16	2057.4	128.59	55.42
SMWSH	16	2108.41	131.77	2.62E-05
REEWS	16	4668.12	291.76	1466.49

ANOVA test result						
Source of Variation	***SS***	***df***	***MS***	***F stat***	***P-value***	***F crit***
Between Groups	351861.89	3	117287.30	308.23	2.13E-36	2.76
Within Groups	22831.32	60	380.52			
Total	374693.12	63				

Tukey-Kramer pairwise between the four algorithms				
Comparison	***Diff***	***Abs. Diff***	***Q crit***	***Is Sig?***
SMWSO vs SCTTEWS	-22.71	22.71	18.22	**YES**
SMWSO vs SMWSH	-25.89	25.89	18.22	**YES**
SMWSO vs REEWS	-185.88	185.88	18.22	**YES**
SCTTEWS vs SMWSH	-3.19	3.19	18.22	NO
SCTTEWS vs REEWS	-163.17	163.17	18.22	**YES**
SMWSH vs REEWS	-159.98	159.98	18.22	**YES**

Algorithm Ranking in terms of Energy saving : SMWSO $\rightarrowtail 1$ | ***SCTTEWS*** $\rightarrowtail 2$ | ***SMWSH*** $\rightarrowtail 2$ | ***REEWS*** $\rightarrowtail 4$

(a) ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for **SIPHT 50**

ANOVA input summary				
Group	***Count***	***Sum***	***Average***	***Variance***
SMWSO	16	4367.86	272.99	1194.02
SCTTEWS	16	4722.25	295.14	1.12
SMWSH	16	4804	300.25	0
REEWS	16	11297.44	706.09	35396.31

ANOVA test result						
Source of Variation	***SS***	***df***	***MS***	***F stat***	***P-value***	***F crit***
Between Groups	2089679.05	3	696559.68	76.14	1.95E-20	2.76
Within Groups	548871.87	60	9147.86			
Total	2638550.92	63				

Tukey-Kramer pairwise between the four algorithms				
Comparison	***Diff***	***Abs. Diff***	***Q crit***	***Is Sig?***
SMWSO vs SCTTEWS	-22.15	22.15	89.36	NO
SMWSO vs SMWSH	-27.26	27.26	89.36	NO
SMWSO vs REEWS	-433.10	433.10	89.36	**YES**
SCTTEWS vs SMWSH	-5.11	5.11	89.36	NO
SCTTEWS vs REEWS	-410.95	410.95	89.36	**YES**
SMWSH vs REEWS	-405.84	405.84	89.36	**YES**

Algorithm Ranking in terms of Energy saving : SMWSO $\rightarrowtail 1$ | ***SCTTEWS*** $\rightarrowtail 1$ | ***SMWSH*** $\rightarrowtail 1$ | ***REEWS*** $\rightarrowtail 4$

(b) ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for **SIPHT 100**

ANOVA input summary				
Group	***Count***	***Sum***	***Average***	***Variance***
SMWSO	16	6817.35	426.08	1145.59
SCTTEWS	16	7686.21	480.39	1.38
SMWSH	16	7809.51	488.09	2.62E-05
REEWS	16	22662.82	1416.423	131227.13

ANOVA test result						
Source of Variation	***SS***	***df***	***MS***	***F stat***	***P-value***	***F crit***
Between Groups	10902391.98	3	3634130.66	109.81	2.47E-24	2.76
Within Groups	1985611.60	60	33093.53			
Total	12888003.58	63				

Tukey-Kramer pairwise between the four algorithms				
Comparison	***Diff***	***Abs. Diff***	***Q crit***	***Is Sig?***
SMWSO vs SCTTEWS	-54.30	54.30	169.96	NO
SMWSO vs SMWSH	-62.01	62.01	169.96	NO
SMWSO vs REEWS	-990.34	990.34	169.96	**YES**
SCTTEWS vs SMWSH	-7.71	7.71	169.96	NO
SCTTEWS vs REEWS	-936.04	936.04	169.96	**YES**
SMWSH vs REEWS	-928.33	928.33	169.96	**YES**

Algorithm Ranking in terms of Energy saving : SMWSO $\rightarrowtail 1$ | ***SCTTEWS*** $\rightarrowtail 1$ | ***SMWSH*** $\rightarrowtail 1$ | ***REEWS*** $\rightarrowtail 4$

(c) ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for **SIPHT 200**

ANOVA input summary				
Group	***Count***	***Sum***	***Average***	***Variance***
SMWSO	16	22340.5	1396.28	1068.52
SCTTEWS	16	23100.45	1443.78	26.91
SMWSH	16	23213.23	1450.83	1.81
REEWS	16	71784.24	4486.51	1220896.02

ANOVA test result						
Source of Variation	***SS***	***df***	***MS***	***F stat***	***P-value***	***F crit***
Between Groups	112113902.3	3	37371300.77	122.33	1.57E-25	2.76
Within Groups	18329898.94	60	305498.31			
Total	130443801.2	63				

Tukey-Kramer pairwise between the four algorithms				
Comparison	***Diff***	***Abs. Diff***	***Q crit***	***Is Sig?***
SMWSO vs SCTTEWS	-47.50	47.50	516.39	NO
SMWSO vs SMWSH	-54.54	54.54	516.39	NO
SMWSO vs REEWS	-3090.23	3090.23	516.39	**YES**
SCTTEWS vs SMWSH	-7.05	7.05	516.39	NO
SCTTEWS vs REEWS	-3042.74	3042.74	516.39	**YES**
SMWSH vs REEWS	-3035.69	3035.69	516.39	**YES**

Algorithm Ranking in terms of Energy saving : SMWSO $\rightarrowtail 1$ | ***SCTTEWS*** $\rightarrowtail 1$ | ***SMWSH*** $\rightarrowtail 1$ | ***REEWS*** $\rightarrowtail 4$

(d) ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for **SIPHT 500**

ANOVA input summary				
Group	***Count***	***Sum***	***Average***	***Variance***
SMWSO	16	46201.01	2887.56	5375.25
SCTTEWS	16	45967.34	2872.96	94.77
SMWSH	16	46008.65	2875.54	27.57
REEWS	16	179212.34	11200.77	25695710.21

ANOVA test result						
Source of Variation	***SS***	***df***	***MS***	***F stat***	***P-value***	***F crit***
Between Groups	831086879.3	3	277028959.8	43.11	5.54E-15	2.76
Within Groups	385518117	60	6425301.95			
Total	1216604996	63				

Tukey-Kramer pairwise between the four algorithms				
Comparison	***Diff***	***Abs. Diff***	***Q crit***	***Is Sig?***
SMWSO vs SCTTEWS	14.60	14.60	2368.22	NO
SMWSO vs SMWSH	12.02	12.02	2368.22	NO
SMWSO vs REEWS	-8313.21	8313.21	2368.22	**YES**
SCTTEWS vs SMWSH	-2.58	2.58	2368.22	NO
SCTTEWS vs REEWS	-8327.81	8327.81	2368.22	**YES**
SMWSH vs REEWS	-8325.23	8325.23	2368.22	**YES**

Algorithm Ranking in terms of Energy saving : SMWSO $\rightarrowtail 1$ | ***SCTTEWS*** $\rightarrowtail 1$ | ***SMWSH*** $\rightarrowtail 1$ | ***REEWS*** $\rightarrowtail 4$

(e) ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for **SIPHT 1000**

Table A.4: ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for LIGO 50, 100, 200, 500 and 10000

ANOVA input summary				
Group	***Count***	***Sum***	***Average***	***Variance***
SMWSO	16	719.68	44.98	2.15E-28
SCTTEWS	16	723.04	45.19	2.15E-28
SMWSH	16	723.04	45.19	2.15E-28
REEWS	16	2621.92	163.87	2676.57

ANOVA test result						
Source of Variation	***SS***	***df***	***MS***	***F stat***	***P-value***	***F crit***
Between Groups	169219.22	3	56406.41	84.30	1.72E-21	2.76
Within Groups	40148.58	60	669.14			
Total	209367.80	63				

Tukey-Kramer pairwise between the four algorithms				
Comparison	***Diff***	***Abs. Diff***	***Q crit***	***Is Sig?***
SMWSO vs SCTTEWS	-0.21	0.21	24.17	NO
SMWSO vs SMWSH	-0.21	0.21	24.17	NO
SMWSO vs REEWS	-118.89	118.89	24.17	**YES**
SCTTEWS vs SMWSH	0	0	24.17	NO
SCTTEWS vs REEWS	-118.68	118.68	24.17	**YES**
SMWSH vs REEWS	-118.68	118.68	24.17	**YES**

Algorithm Ranking in terms of Energy saving : SMWSO $\rightarrowtail 1$ | ***SCTTEWS*** $\rightarrowtail 1$ | ***SMWSH*** $\rightarrowtail 1$ | ***REEWS*** $\rightarrowtail 4$

(a) ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for **LIGO 50**

ANOVA input summary				
Group	***Count***	***Sum***	***Average***	***Variance***
SMWSO	16	2029.12	126.82	1.9387E-27
SCTTEWS	16	2593.28	162.08	8.61646E-28
SMWSH	16	2593.28	162.08	8.61646E-28
REEWS	16	7188.92	449.31	18688.93

ANOVA test result						
Source of Variation	***SS***	***df***	***MS***	***F stat***	***P-value***	***F crit***
Between Groups	1085935.98	3	361978.66	77.47	1.29E-20	2.76
Within Groups	280334.00	60	4672.23			
Total	1366269.99	63				

Tukey-Kramer pairwise between the four algorithms				
Comparison	***Diff***	***Abs. Diff***	***Q crit***	***Is Sig?***
SMWSO vs SCTTEWS	-35.26	35.26	63.86	NO
SMWSO vs SMWSH	-35.26	35.26	63.86	NO
SMWSO vs REEWS	-322.49	322.49	63.86	**YES**
SCTTEWS vs SMWSH	0	0	63.86	NO
SCTTEWS vs REEWS	-287.23	287.23	63.86	**YES**
SMWSH vs REEWS	-287.23	287.23	63.86	**YES**

Algorithm Ranking in terms of Energy saving : SMWSO $\rightarrowtail 1$ | ***SCTTEWS*** $\rightarrowtail 1$ | ***SMWSH*** $\rightarrowtail 1$ | ***REEWS*** $\rightarrowtail 4$

(b) ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for **LIGO 100**

ANOVA input summary				
Group	***Count***	***Sum***	***Average***	***Variance***
SMWSO	16	4736.64	296.04	0
SCTTEWS	16	6020.64	376.29	0
SMWSH	16	5839.2	364.95	1.38E-26
REEWS	16	16694.68	1043.4175	111879.77

ANOVA test result						
Source of Variation	***SS***	***df***	***MS***	***F stat***	***P-value***	***F crit***
Between Groups	5901070.49	3	1967023.50	70.33	1.26E-19	2.76
Within Groups	1678196.63	60	27969.94			
Total	7579267.123	63				

Tukey-Kramer pairwise between the four algorithms				
Comparison	***Diff***	***Abs. Diff***	***Q crit***	***Is Sig?***
SMWSO vs SCTTEWS	-80.25	80.25	156.25	NO
SMWSO vs SMWSH	-68.91	68.91	156.25	NO
SMWSO vs REEWS	-747.38	747.38	156.25	**YES**
SCTTEWS vs SMWSH	11.34	11.34	156.25	NO
SCTTEWS vs REEWS	-667.13	667.13	156.25	**YES**
SMWSH vs REEWS	-678.47	678.47	156.25	**YES**

Algorithm Ranking in terms of Energy saving : SMWSO $\rightarrowtail 1$ | ***SCTTEWS*** $\rightarrowtail 1$ | ***SMWSH*** $\rightarrowtail 1$ | ***REEWS*** $\rightarrowtail 4$

(c) ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for **LIGO 200**

ANOVA input summary				
Group	***Count***	***Sum***	***Average***	***Variance***
SMWSO	16	29398.4	1837.4	2.20E-25
SCTTEWS	16	18780.8	1173.8	5.51E-26
SMWSH	16	15693.92	980.87	1.24E-25
REEWS	16	55628.8	3476.8	1807613.57

ANOVA test result						
Source of Variation	***SS***	***df***	***MS***	***F stat***	***P-value***	***F crit***
Between Groups	61729354.04	3	20576451.35	45.53	1.81E-15	2.76
Within Groups	27114203.59	60	451903.39			
Total	88843557.63	63				

Tukey-Kramer pairwise between the four algorithms				
Comparison	***Diff***	***Abs. Diff***	***Q crit***	***Is Sig?***
SMWSO vs SCTTEWS	663.6	663.6	628.05	**YES**
SMWSO vs SMWSH	856.53	856.53	628.05	**YES**
SMWSO vs REEWS	-1639.4	1639.4	628.05	**YES**
SCTTEWS vs SMWSH	192.93	192.93	628.05	NO
SCTTEWS vs REEWS	-2303	2303	628.05	**YES**
SMWSH vs REEWS	-2495.93	2495.93	628.05	**YES**

Algorithm Ranking in terms of Energy saving : SMWSO $\rightarrowtail 3$ | ***SCTTEWS*** $\rightarrowtail 1$ | ***SMWSH*** $\rightarrowtail 1$ | ***REEWS*** $\rightarrowtail 4$

(d) ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for **LIGO 500**

ANOVA input summary				
Group	***Count***	***Sum***	***Average***	***Variance***
SMWSO	16	20234.08	1264.63	2.20E-25
SCTTEWS	16	39513.44	2469.59	0
SMWSH	16	39513.44	2469.59	0
REEWS	16	82536.4	5158.52	1536735.65

ANOVA test result						
Source of Variation	***SS***	***df***	***MS***	***F stat***	***P-value***	***F crit***
Between Groups	130108073.4	3	43369357.79	112.89	1.23E-24	2.76
Within Groups	23051034.81	60	384183.91			
Total	153159108.2	63				

Tukey-Kramer pairwise between the four algorithms				
Comparison	***Diff***	***Abs. Diff***	***Q crit***	***Is Sig?***
SMWSO vs SCTTEWS	-1204.96	1204.96	579.09	**YES**
SMWSO vs SMWSH	-1204.96	1204.96	579.09	**YES**
SMWSO vs REEWS	-3893.89	3893.89	579.09	**YES**
SCTTEWS vs SMWSH	0	0	579.09	NO
SCTTEWS vs REEWS	-2688.93	2688.93	579.09	**YES**
SMWSH vs REEWS	-2688.93	2688.93	579.09	**YES**

Algorithm Ranking in terms of Energy saving : SMWSO $\rightarrowtail 1$ | ***SCTTEWS*** $\rightarrowtail 2$ | ***SMWSH*** $\rightarrowtail 2$ | ***REEWS*** $\rightarrowtail 4$

(e) ANOVA test along with Tukey-Kramer pairwise tests comparing the Energy Consumption of SCTTEWS, SMWSO and SMWSH vs REEWS for **LIGO 1000**

CHAPTER

B

Appendix B

B.1 Scientific Publications

In this section, scientific publications of NDAMLABIN MBOULA Jean Etienne are provided.

B.1.1 Scientific publications related to this thesis

The following publications are directly related to the content of this thesis:

[1]- **J. E. Ndamlabin Mboula**, V. C. Kamla, and C. Tayou Djamegni: *Cost-time trade-off efficient workflow scheduling in cloud*. Simulation Modelling Practice and Theory 103 (2020): 102107. Elsevier.

[2]- **J. E. Ndamlabin Mboula**, V. C. Kamla, and C. Tayou Djamegni: *Dynamic provisioning with structure inspired selection and limitation of VMs based cost-time efficient workflow scheduling in the cloud*. Cluster Computing 24 (2021): 2697-2721, Springer.

B.1.2 Other publications

The following publication is not directly related to the content of this thesis:

[3]- V. C. Kamla, **J. E. Ndamlabin Mboula**, J. S. Wouansi, and C. Tayou Djamegni: *Grid's Acquaintance-Based Multiagent Model of Distributed Meta-Scheduling*. In Signal-Image Technology & Internet-Based Systems (SITIS), 2016 12th International Conference on (pp. 295-301). IEEE.