THE UNIVERSITY OF
SYDNEY

DOCTORAL THESIS

Scientific Workflow Scheduling for Cloud Computing Environments

---

Author:
ISRAEL CASAS

Advisor:
PROF. ALBERT Y. ZOMAYA

A thesis submitted in fulfilment of the requirements
for the degree of ***Doctor of Philosophy***
in the

CENTRE FOR DISTRIBUTED & HIGH PERFORMANCE COMPUTING
FACULTY OF ENGINEERING AND INFORMATION TECHNOLOGIES

The University of Sydney

May 2017

Declaration of Authorship

I, **Israel Casas**, certify that:

- This thesis comprises only my original work and has not been submitted in any form for a degree at any university.

- The intellectual content of this thesis is the product of my own work and that all the assistance received in preparing this thesis and sources have been acknowledged, and a list of references is given.

- This thesis contains material published in [1-3]. These are Chapters 4 5 and 6 . For the three articles I designed the study, ran experiments and wrote the manuscripts.

_____

Israel Casas

May 4, 2017

As supervisor for the candidature upon which this thesis is based, I can confirm that the authorship attribution statements above are correct.

_____

Professor Albert Y. Zomaya
Principal Advisor
May 4, 2017

Dedicated to my parents
Cecilia and Javier

Acknowledgements

I feel enormously thankful to have worked with exceptional professors. Foremost, thanks to Professor Albert Y. Zomaya who directed and gave a strong base to my doctorate. Special thanks to Dr. Javid Taheri for his priceless support, he was a key figure to build the structure of my research. In deep thanks I feel to Dr. Rajiv Ranjan for an invaluable support on the fundamental moments on this research. Similarly, I feel thankful to Joanne Allison for her support on the edition of this document. Special thanks to CSIRO (Commonwealth Scientific and Industrial Research Organization) and CONACYT (Consejo Nacional de Ciencia y Tecnología) for their economic support to produce this research.

Besides the academic field, I would like to thanks to the Long Family for have given me a priceless support, memorable times, adventures, and more – Adam, Yvonne, Sean, Cayla, John, Garry, Margaret, Rebecca, Kyle, Dean, Jordan, Christian, Aaron, Elvia and Ernie – for life I will be thankful with you all. Special thanks to the Christian family, it is incredible how much a friendship can achieve – Michael, Lyn, Rayner, Jeremy, Tanya, Gavin and Kelly – thank you all. Thanks to my sister Samantha, for every call, every message and every word, it really made a difference to me. Similarly, I thank to Shaghayegh Sharif Nabavi for her unconditional and fraternal friendship.

Finally and most importantly, I would like to thanks to my parents, Cecilia and Javier for all the love they have given me.

Abstract

Scientists use workflow applications to automate their experiments. To run these workflows on computer systems, each of their tasks must be scheduled to a computational resource. The scheduling of a task consists of assigning it to a resource in order to fulfill a final goal such as minimizing total workflow execution time. For this reason, workflow scheduling plays a crucial role in efficiently running experiments. Workflows often have many discrete tasks and the number of different task distributions possible and consequent time required to evaluate each configuration quickly becomes prohibitively large. For these reasons, the scheduling of workflows is considered to be an NP-hard problem, i.e. a problem not solvable within polynomial time with current resources.

A proper solution to the scheduling problem requires the analysis of tasks and resources, production of an accurate environment model and, most importantly, the adaptation of optimization techniques. To date, different solutions have been developed to schedule complex applications on computing systems. Approaches thus far fail in (1) providing a deep analysis of task interdependencies to fully exploit parallelism, (2) incrementing computer system utilization, and (3) adapting the number of resources to run each workflow. This study is a major step toward solving the scheduling problem by not only addressing these issues but also optimizing the runtime and reducing monetary cost, two of the most important variables.

To achieve these goals, this study proposes three scheduling algorithms capable of answering key issues to solve the scheduling problem. Firstly, it unveils BaRRS, a scheduling solution that exploits parallelism and optimizes runtime and monetary cost. BaRRS is also capable of producing configurations with high system utilization values, an important characteristic of running workflows on public cloud systems. Secondly, it proposes GA-ETI, a scheduler capable of returning the number of resources that a given workflow requires for execution. GA-ETI utilizes the capabilities of the genetic evolution to overcome current scheduling algorithm deficiencies in terms of execution time and cost. Finally, it describes PSO-DS, a scheduler based on particle swarm optimization to efficiently schedule large workflows. PSO-DS is able to converge to a final solution without significantly adding time overheads.

To test the performance of BaRRS, GA-ETI and PSO-DS, they are compared to the current state of the art scientific workflow schedulers. Experiments include a test bed based on the VMware-vSphere and a cloud environment built on the Krypton Quattro R6010 server. To test the algorithms, five well-known benchmarks are selected that represent different scientific applications. The experiments found the novel algorithms solutions substantially improve efficiency, reducing makespan by 11% to 78%. This represents a significant improvement and major contribution to the field. The proposed frameworks open a path for building a complete system that encompasses the capabilities of a workflow manager, scheduler, and a cloud resource broker in order to offer scientists a single tool to run computationally intensive applications.

Contents

List of figures

List of tables

1 Introduction

1.1 Preliminaries

Computer technology is a vital tool for scientific investigations. Workflows, the connection between scientists and computer systems, are a collection of computational tasks organized to accomplish a composite assignment as in climate modelling, genome sequencing, seismic analysis and oil exploration. Scientific workflows include hundreds or thousands of computational tasks which are interconnected following different dependency patterns. Workflow tasks habitually require large input data files and/or perform an extraordinary number of instructions. These factors provoke scientific workflows to produce a high number of combinations to distribute their tasks on computer resources. As a consequence, the process to select the optimal distribution becomes a complicated problem.

To cope with this problem, computational systems have a scheduling stage. During this stage, workflows are discerned in order to discover the best distribution of their tasks to computational resources. Formally expressed, scientific workflow scheduling is the analysis of application structures to optimally assign tasks to computational resources based on application characteristics and resource availability. The aim of workflow schedulers is to produce a satisfactory solution in a relatively short time.

Nevertheless, producing an optimal scheduling configuration becomes a serious problem as the number of tasks increments. Also two of the most important variables, i.e. execution time and monetary cost, are two conflicting objectives during optimization. On one side, optimal execution times converge with solutions employing the fastest and most expensive computer resources. On the other hand, a full optimization of monetary cost leads to poor performance in terms of execution time. For the aforementioned issues, the scheduling of workflows is classified as an NP-complete problem, i.e. a problem that cannot be solved within polynomial time using current computing systems.

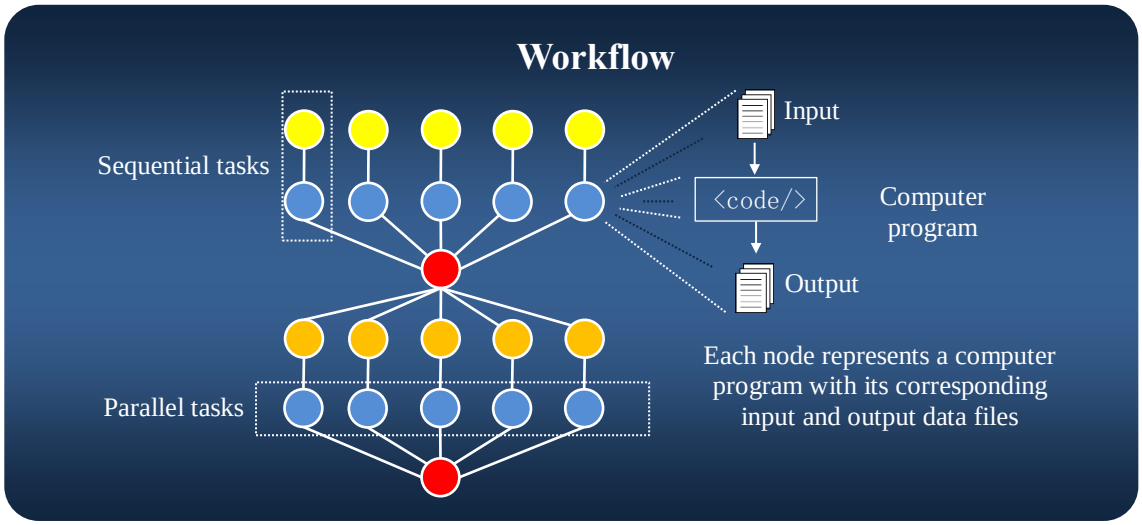


Figure 1. Example of a workflow.

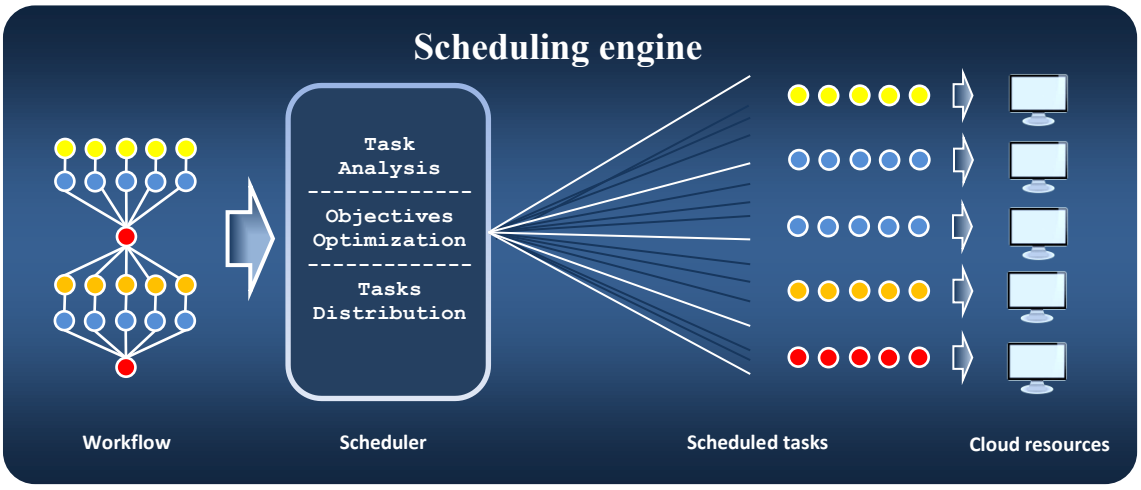


Figure 2. Scheduler interaction with workflow and cloud resources.

Last but not least, the performance of schedulers depends directly on the targeted computing system. Nowadays, different computer systems have the required capabilities to execute these applications; nevertheless, cloud computing has the most attractive environment to run scientific workflows due to five main characteristics. Firstly, cloud computing systems have extraordinary amounts of computing power and massive data storage capacity. Secondly, contrary to grid computing systems, every person and/or institution can access cloud resources without any affiliation. Thirdly, cloud systems prevent the need for their users to invest in costly systems, such as supercomputers. Fourthly, contrary to cluster computing systems, cloud customers can scale up/down the number of resources. Finally, cloud customers can have immediate access to computing resources while supercomputing system users are usually required to wait for weeks to have access to resources.

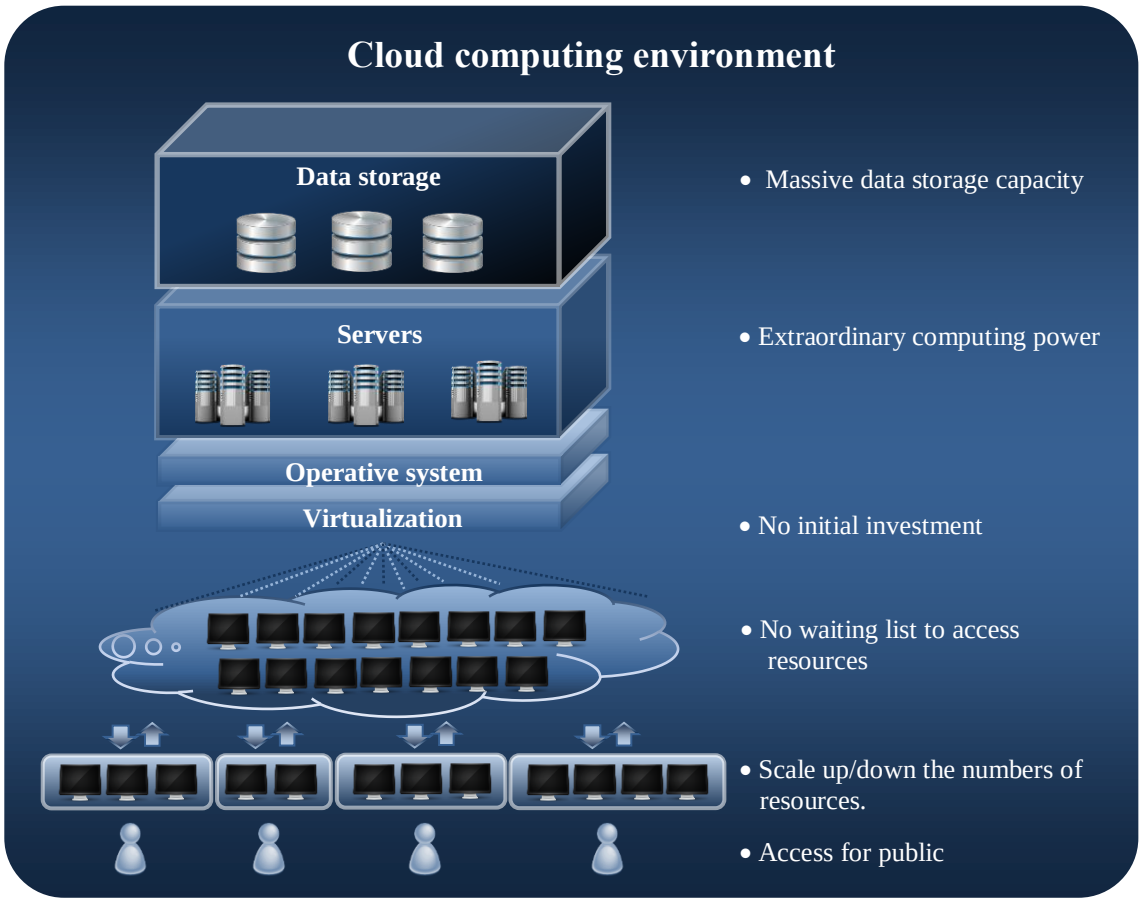


Figure 3. Cloud computing system.

However, cloud computing systems increase the difficulty of scheduling scientific workflows because of their massive pools of resource. For this reason, scheduling frameworks have been a focus of attention for researchers in the area of information technology. Scheduling frameworks are of great importance to cloud computing systems. They are a key element to increment system efficiency. They are also a lead driver for cloud systems in being the main computer source that the scientific community employs to produce discoveries in genomics, physics and medicine. Similarly, optimal scheduling frameworks lead to proficient usage of cloud resources which in turn leads to energy savings, a crucial concern for environmental health.

The main challenges of the scheduling problem are enumerated as follows:

1. Optimize the two essential but conflicting objectives, i.e. Runtime and monetary cost.
2. Maintain at all times a balanced task distribution among computers to increment system utilization.
3. Exploit parallelism and discover alternatives that converge to optimal solutions.
4. Adapt the necessary optimization theories to deliver superior results and then compare these with current solutions.
5. Define and calculate the optimal number of resources to run a particular workflow.
6. Deliver a solution with a low complexity in order to enable scalability and the capability to analyze large workflow sizes.

For the aforementioned reasons, this thesis investigates the scheduling problem to execute workflows in cloud computing environments. The study observed that current workflows and the vast computational offers available are challenging the workflow scheduling analysis with which current scheduling algorithms are not fully able to cope as yet. For the above mentioned reasons, the study undertook in depth research to understand the capabilities of Genetic Evolution and Particle Swarming to solve the workflow scheduling problem. Exploration of these capabilities can lead to unprecedented improvements on scheduling performance.

1.2 Motivation

Even though several scheduling concerns have been extensively investigated, such as the uncertainties produced by system failures, resource scalability, computer heterogeneity, budget restrictions and deadline constraints among others, there are still some other concerns that have not attracted the attention they deserve. After a deep analysis of such factors, this thesis has identified three specific issues preventing scheduling algorithms from incrementing their efficiency. These key factors are described below.

1.2.1 Analysis of Task Interdependencies to Reduce File Transfer and/or Exploit Data Replication

Task interdependencies must obtain similar attention as task processing times during scheduling analysis. This requirement emerged after an in depth analysis of scientific applications as in [4], for which this study found that dependencies between tasks offer an important opportunity to reduce file transfers where parallelism doesn't offer improvement. Parallelization, i.e. the execution of tasks at the same time, is an important factor to incrementing performance when running applications over distributed systems. Even though cloud environments are ruled by an economic model that influences scheduling decisions, the scheduler should have flexibility in sacrificing parallelization in order to fully use resources to increment performance. Moreover, a great number of workflow schedulers exploit parallelism whenever possible without considering the trade off with the rest of variables as a monetary cost.

1.2.2 Increment System Utilization in Public Cloud Environments

System utilization has a great impact on users and cloud provider objectives. From the user's side, they generally aim to execute applications at the lowest cost possible. On the other side, public cloud providers intend to maintain an efficient quality of a service for every user. Load balancing offers an opportunity to increment system utilization by evenly distributing workloads among computing resources. Load balancing can be implemented during decision making or it can run at periodic intervals during a scheduling process, i.e. workflow balancing strategies in parallel machine scheduling. File transfer time reduction is another important factor to incrementally improve system utilization. Even then, a scheduler

needs to produce a trade-off between the number of replica files and task processing time to choose an optimal task distribution in order to reduce total execution time. For the aforementioned reasons, load balancing and data replication decisions must be embedded in the scheduling engine to ensure the efficient distribution of load to increment system utilization.

1.2.3 Adapt the Number of VMs to Workflow Requirements

Most scheduling algorithms use a fixed number of VMs to execute schedule tasks with only a few exceptions where the number of VMs follows a simplistic criteria as in [5-14]. In [5-9], authors drive the number of VMs selection by an equilibrium between a monetary cost constraint and task computational demands while the scheduling mechanisms in [10-14] require minor modifications for allowing number of VM selection. We observe that a number of VMs are directly related to the workflow size and specific parameters. The number of tasks in a workflow may be seen as the most important parameter to select the VM pool size, even though parallelism has a stronger influence in selecting the number of VMs to run a particular application since it dictates how many tasks can be executed at the same time. To the best of our knowledge, this fact hasn't been analyzed in previous works. This study shows that adapting the number of VMs to execute a given workflow has a decisive impact on the performance of the execution of the application on cloud environments.

1.3 Research Objectives

A high-performance algorithm, in this context, is a mechanism capable of producing higher performance results compared to current workflow manager systems in terms of execution time and monetary cost. The fundamental objective of this thesis is to design high performance algorithms to solve the scheduling problem while considering task interdependencies, the need to balance loads among cloud resources and the selection of VM pool sizes. Task interdependencies considered in this thesis are: pipeline, data distribution, data aggregation and data redistribution [4]. Pipeline structures joins tasks serially, data distribution are group of tasks requesting a single set of input data files, data aggregation are tasks requesting data files from at least two other parent tasks, and data redistribution combines interdependencies (2) and (3) producing inputs for multiple tasks. This thesis

engages the aforementioned concepts by conducting a specific study that has three main objectives:

1. Analyze the key issues that have a strong influence on workflow scheduling in order to: select a scheduling option based on particular workflow task dependencies, maintain computer workload at all times and select a number of VMs that allows minimal execution time and monetary cost compared with current scheduling mechanisms.
2. Design new algorithms based on genetic evolution, particle swarming and system utilization enhancers to incorporate the aforementioned requirements in the process of assigning workflow tasks to cloud resources.
3. Validate the efficiency of the proposed algorithms in a controlled environment employing benchmarks representing current scientific applications.

Therefore, this thesis performed an in depth investigation into genetic evolution and particle swarm optimization algorithms to develop novel scheduling algorithms as well as to increment our understanding of the key characteristics to adapt such evolutionary algorithms into the scheduling problem. This study proposes three novel approaches to address the scheduling problem for an optimal execution of scientific workflows on cloud computing environments.

1.4 Contributions and Research Methodology

To achieve the proposed thesis objectives, this study engages in a triple stage methodology as shown in Figure 4. BaRRS, GA-ETI and PSO-DS Framework: firstly, design a scheduling engine highlighting the importance of system utilization; secondly, adapt a genetic algorithm to consider a higher number of solutions; and thirdly, adapt the PSO mechanism to include all the above features delivering an algorithm with a low complexity. Characteristics of each stage are described below in Subsections 1.4.1 – 1.4.3. The proposed scheduling mechanisms are then evaluated by conducting experiments on our private cloud environment capable of generating AWS [15] instances such as the t2.micro, t2.small, t2.medium and t2.large. The experimentation uses different benchmarks representing up-to-date applications from different scientific areas.

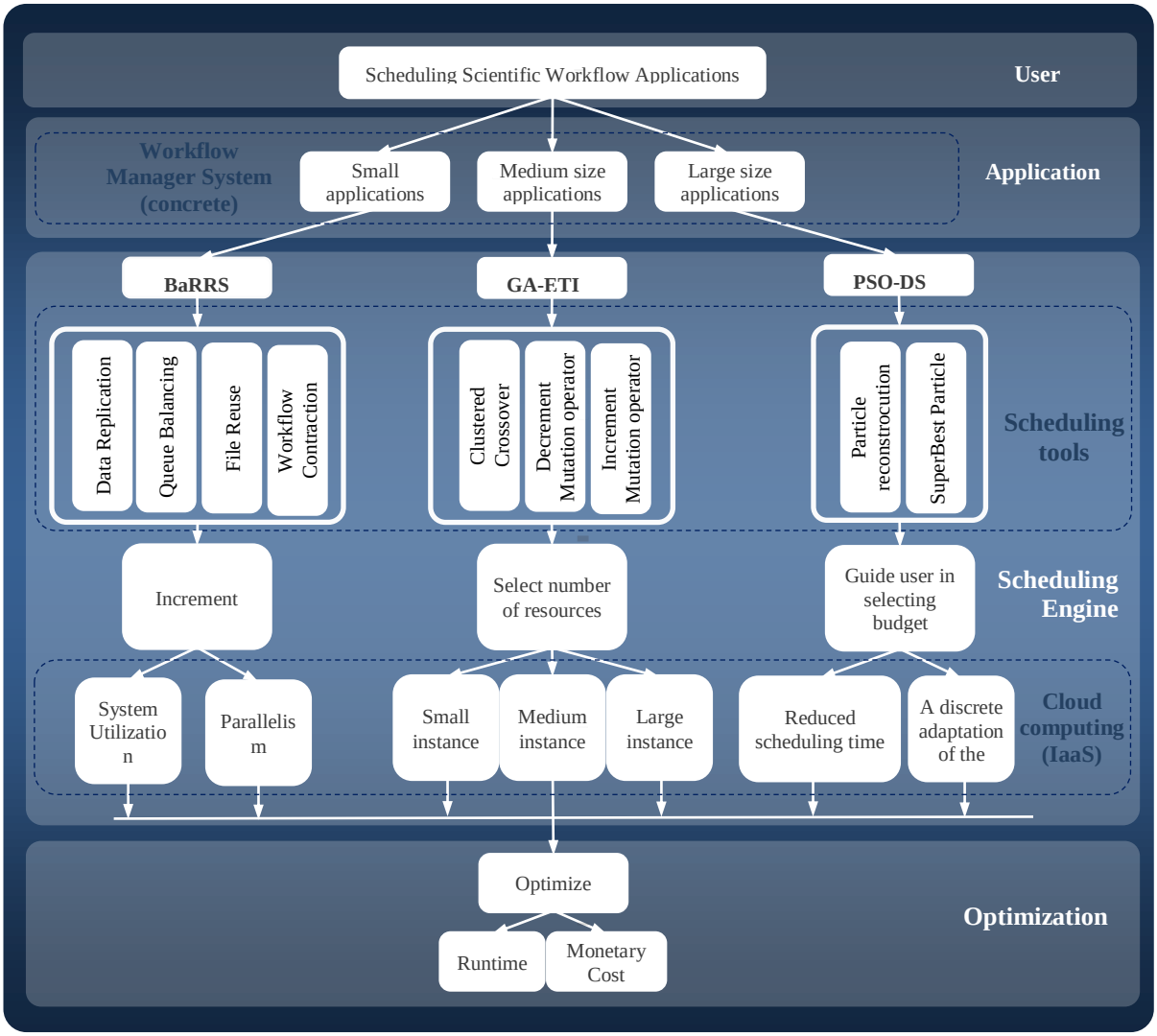


Figure 4. BaRRS, GA-ETI and PSO-DS Framework

Table 1. Granularity Level

Execution Time (ExT)	Transfer Time (TT)*	Granularity (ExT/TT)
355	304	1.167763
10	2.85816	3.498754
63	6.4	9.84375
10	0.0636	157.2327
9635	23.712	406.3343
1344	2.064	651.1628
460	0.592	777.027
3333	2.08	1602.404
13	0.00024	54166.67
975	0.00060	1625000

*Considering a bandwidth of 1Mbps

This work targets the scheduling of scientific workflow with different number of tasks. Where each task requires computing tasks and transferring data files. For specifying

the granularity levels this thesis targeted, Table 1 presents ten examples of data/computer intensive tasks, targeted by the proposed algorithms, exhibiting different granularity values. For instance, the first task exhibits the lowest granularity level and refers to a data intensive task from an application analysing seismic hazards while the last task shows the greatest granularity level and refers to a computer intensive task encoding genes in a bioinformatics application.

1.4.1 Balanced and File Reuse-Replication Scheduling (BaRRS)

This study introduces BaRRS, a novel algorithm that encompasses three scheduling mechanisms with the objective of optimizing runtime and monetary cost. Firstly, BaRRS customizes data reuse techniques to reduce data transfers by assigning parallel tasks to the same VMs (whenever they reduce execution time). Secondly, the algorithm includes file replication to complement data reuse decisions to duplicate files whenever parallel execution optimizes runtime. Finally, BaRRS balances workloads among resources to increment system utilization. The main contributions of BaRRS are:

1. A solution to the problem to optimize the two essential, yet conflicting, objectives in workflow scheduling: runtime and monetary cost.
2. Introduces a triple scheduling mechanism based on load balancing, file reuse and data replication to exploit parallelism and to increment system utilization.
3. Additionally, it computes a trade-off exhibiting the different levels to optimize the selected objectives.

BaRRS bases its scheduling decisions on the analysis of task dependencies, file sizes, task execution times, and network bandwidth, as well as the underlying VMs' characteristics.

1.4.2 Genetic Algorithm with Efficient Tune-In of Resources (GA-ETI)

This study proposes the GA-ETI by incorporating the BaRRS techniques into an adaptation of the genetic algorithm to select the optimal number of resources and to minimize makespan and monetary cost values. A genetic evolution based scheduler provides the following distinctive elements: (1) considers a population of solutions rather than building a unique solution dismissing scheduling configurations that can lead to an optimal result, (2)

has a population of solutions that evolve to produce an optimal (or close to optimal) solution considering a wide range of solution spaces and (3) is a random guided process. More importantly, GA-ETI holds specific characteristics that obtain superior results compared to other GA based schedulers. The main contributions of GA-ETI include:

1. A solution to the problem to select the number of resources to execute workflows in cloud computing systems.

2. A scheduling framework based on the capabilities of the genetic evolution theory to provide superior results in terms of execution time and monetary cost over up-to-date schedulers.

3. Additionally, this study concurrently reduces the characteristic randomness of the genetic algorithm leading the GA-ETI to converge to a final solution with fewer generations when compared with similar adaptations of the GA.

The GA-ETI mutation operator injects new VMs into a population to warranty diversity among chromosomes. GA-ETI aims to consider a wider range of scheduling options in order to produce an optimal (or close to optimal) final solution.

1.4.3 Particle Swarm Optimization with Discrete Adaptation and a Featured SuperBEST (PSO-DS)

This study discloses the PSO-DS by embracing the capabilities of the BaRRS and GA-ETI to manage large workflow size with a satisfactory scheduling overhead time. Scheduling algorithms based on the PSO generally: (1) are relatively straightforward to implement; (2) require a low number of variables to set up compared to genetic algorithms; and (3) are capable of considering a space of solutions wide enough to provide a global optimal result. The main contributions of the PSO-DS are:

1. A solution to the problem to schedule large size workflows in cloud computing systems.

2. Extract and adapt the capabilities of the PSO to build a scheduler capable of executing applications on a record time.

3. Additionally, this study produces an enhanced PSO particle reconstruction to produce scheduling configurations not achievable with existing approaches and introduces a Super Particle that shortens the PSO search time.

The proposed algorithm produces an adaptation of the original discrete particle swarm optimization that is introduced into scheduling algorithms, where particles represent complete scheduling configurations aiming to improve their evaluation value in terms of execution time and monetary cost.

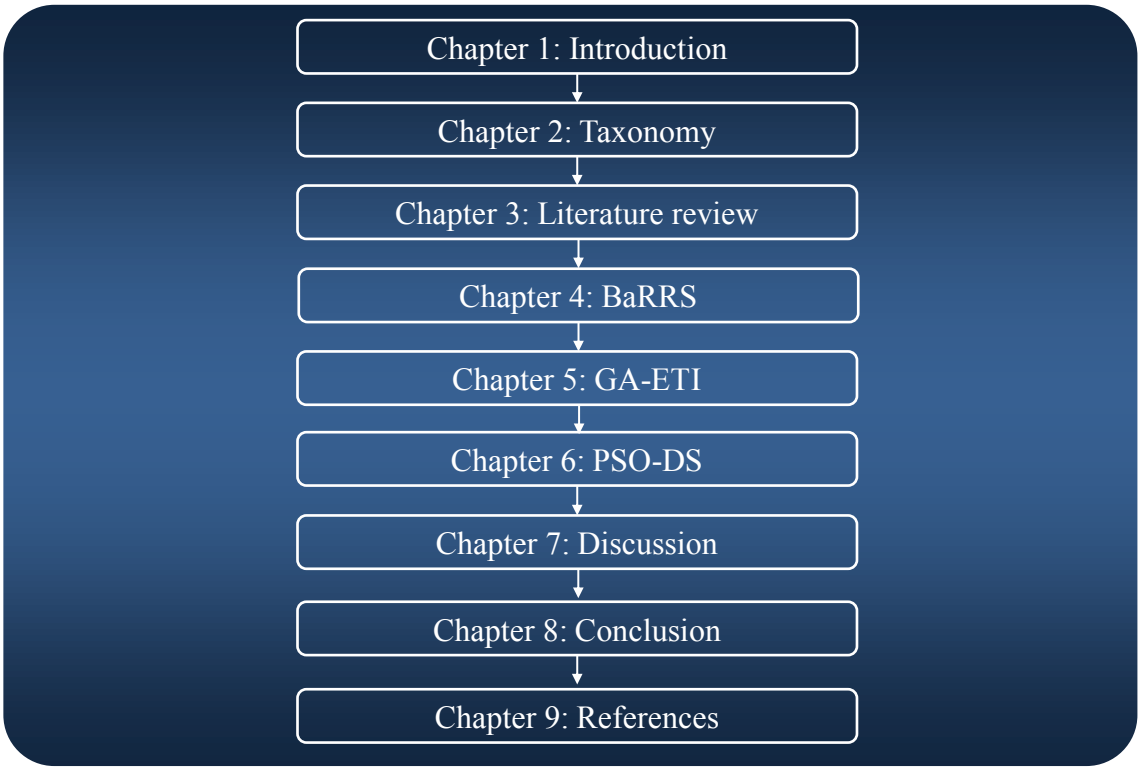


Figure 5. Thesis outline

1.5 Thesis Organization

The thesis outline follows a course as presented in Figure 5. Chapter 2 presents the taxonomy of workflow applications and schedulers then it presents the literature review on the scheduling algorithms. Chapter 4 presents BaRRS, a triple scheduling mechanism to solve the scheduling problem. Chapter 5 explores GA-ETI, a scheduler based on the capabilities of the genetic algorithm. Chapter 6 introduces PSO-DS, a PSO scheduler capable of scheduling large workflow sizes. Chapter 7 concludes with a summary of the relevant issues presented during experimentations, contributions, and discusses future work as well.

1.6 List of publications

Following is a list of publications achieved during the realization of this study. The core of Journal 1 is presented in Chapter 4. Similarly, the main contribution of Journal 2 is analyzed in Chapter 5. Finally, the work of Journal 3 is analyzed in Chapter 6.

Journals

1. **I. Casas**, J. Taheri, R. Ranjan, L. Wang, and A. Y. Zomaya, "A balanced scheduler with data reuse and replication for scientific workflows in cloud computing systems", *Future Generation Computer Systems*, 2016.

2. **I. Casas**, J. Taheri, R. Ranjan, L. Wang, and A. Y. Zomaya, "GA-ETI: An Enhanced Genetic Algorithm for the Scheduling of Scientific Workflows in Cloud Environments", *Journal of Computational Science*, 2016.

3. **I. Casas**, J. Taheri, R. Ranjan, L. Wang, and A. Y. Zomaya, "PSO-DS: A Scheduling Engine for Scientific Workflow Managers", Journal of Supercomputing, 2016. (Under revision)

2 Taxonomy

2.1 Preliminaries

This chapter presents taxonomy of workflow applications, scheduling engine workflow manager systems and cloud computing environments. Firstly, Section 2.2 presents the taxonomy of workflows to understand their application structure, and then Section 2.3 explores the taxonomy of the scheduling engines focusing on their architectures, optimization, and dynamism and problem types. Then, Section 2.4 3.2 presents the taxonomy of a workflow manager system. Finally, Section 2.5 explores the characteristics of a cloud computing system.

2.2 Workflow Application

As expressed in Figure 6, workflow taxonomy consists of four main elements: structure, domain, specification and composition. The following paragraphs describe each of these elements.

Workflow structure

Workflows have a fixed number of computational tasks. A computational task, also named job, is a set of instructions requiring a set of input files. Tasks may exhibit interdependencies; as a consequence, applications present two structure types: BoT and DAGs. Bag of Tasks (BoT) are applications with parallel tasks with no interdependencies between the tasks. Directed Acyclic Graphs (DAGs) are applications with tasks connected by edges (data files). Task weight expresses processing time while edges denote file size or file transfer cost. Workflows are commonly modeled as DAGs where nodes contain a task with its required file or files set [9, 16-18].

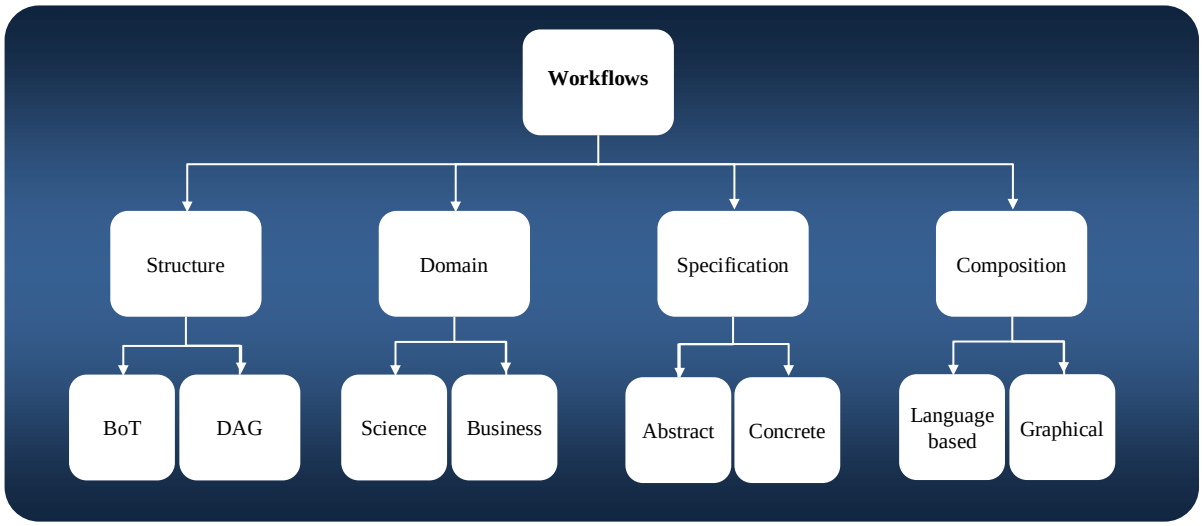


Figure 6. Taxonomy of Scientific Application.

Workflow domain

Applications cover two main domains: science and business. On one side, science applications such as bioinformatics, data mining, high-energy physics, astronomy and neuroscience benefit from cloud services to store, retrieve and run experiments. On the other hand, business applications such as in economy, forecasting and oil exploration take advantage of the scalability and performance of cloud computing services to extend their computing power for fixed periods of time.

Workflow specification

Workflows can have an abstract or concrete description. Abstract workflow description does not include low-level implementation details such as assigning a given task to a particular resource. Consequently, tasks in an abstract workflow model are portable to other computational systems. In contrast, concrete workflow descriptions fix tasks to specific resources for their execution.

Workflow composition

Application owners can use syntax or graphics to express a workflow. Language-based describes workflows using syntax such as XML (Extensive Markup Language). Graphical expressed workflows require low-level details; hence users focus on a higher abstraction of the application. Unified Modeling Language (UML) is a well know graphic modeling system for workflows.

2.3 Scheduling Engine

As expressed in Figure 7, the scheduling engine taxonomy consists of four main fundamentals: architecture, optimization objectives, dynamism and problem type. The following paragraphs describe each of these elements.

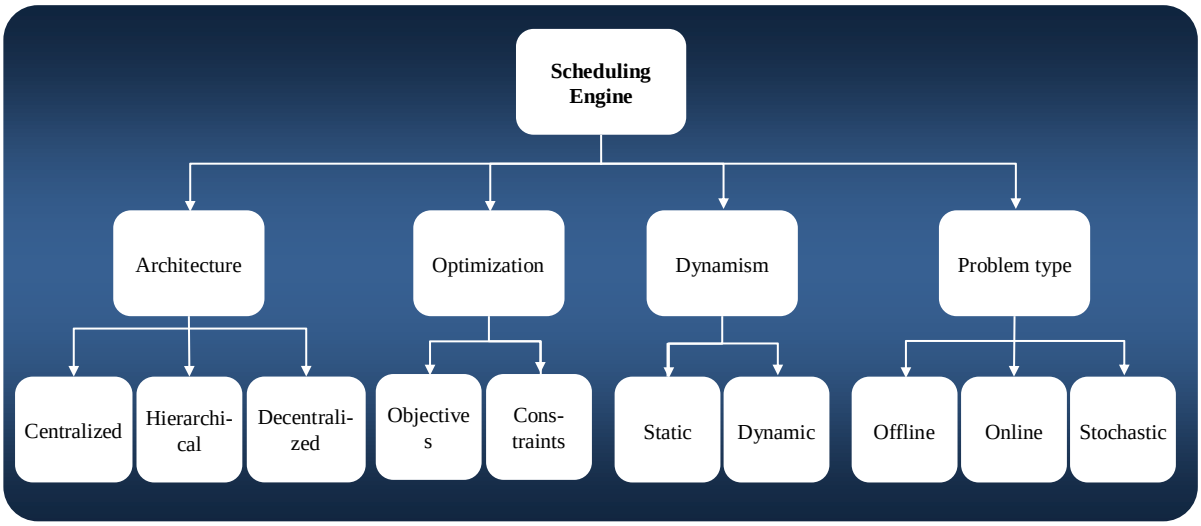


Figure 7. Taxonomy of Scheduling Engine.

Scheduling Architecture

Architecture refers to the origin of scheduling decisions. Schedulers with a centralized architecture produce all their decisions from a central controller. These schedulers keep all information regarding application execution and resource status. The main feature of central schedulers is their high simplicity to implement and deploy. Hierarchical schedulers have a central director and lower-level schedulers. The central director handles application execution while low-level schedulers handle individual tasks. Decentralized schedulers contain different scheduling units; each one manages a fixed number of tasks. Decentralized schedulers are excellent tools for applications requiring scalability. Examples of different scheduling architectures appear in [19-22]

Scheduling Optimization

Scheduling engines present two optimization criteria to enhance the execution of applications. On one side, optimization based on objectives has the goal of meeting the best possible value for a given criterion. Regarding the nature of an objective, optimization can maximize or minimize the selected objective. For instance, monetary cost optimization targets a minimization of cost values. On the other side, scheduling constraints restrict particular values such as a deadline or a budget limitation.

Scheduling Dynamism

Static and dynamic scheduling refers to time between the moment a task is scheduled and its actual execution. A static scheduler produces a full plan prior to workflow execution; it enables a deep analysis producing superior results. Dynamic schedulers delay decision-making as long as possible, producing scheduling decisions individually for each task. As a consequence, dynamic schedulers are not able to analyze the whole application at once.

Scheduling Problem Type

The model of a scheduling problem depends on the nature of the application and information available (such as task processing time, size of application and file sizes). In some cases, schedulers have all the workflow information required. In other cases, schedulers do not have any information about tasks, leaving them in a complex situation in relation to their decision making. This taxonomy classifies scheduling problems into three types:

Offline Scheduling is the perfect scenario to produce a deterministic scheduling. In an offline scenario, the scheduler has all the required information prior to any decision making. This information includes job execution and file transfer times. Most schedulers in this area focus on polynomial time algorithms, complexity proofs, heuristics and worst case analysis.

Online Scheduling is the no-information case. The decision making is based on least information or even with no information known previously. The scheduler obtains information gradually as the application executes. Whether execution time is given at arrival or not, online schedulers have a defined objective to minimize the time related to making the best decisions to achieve its goals.

Stochastic Scheduling is the distributional information case. Stochastic schedulers manage decisions based on deterministic and distributional information. For instance, a number of jobs may be fixed and known in advance while job execution time is obtained from a probability distribution. Actual execution time is only known at the end of job execution. Stochastic schedulers have to determine policies to minimize objectives in a stochastic sense. In [23], the author identified static and preemptive stochastic scheduling policies. Static policies specify actions prior to the execution process without deviating from them if more information becomes available. Preemptive dynamic policies take decisions at any time as information becomes available.

Stochastic scheduling problems can have a static model under certain circumstances. For instance, on applications with job processing times falling in an arbitrary distribution, a stochastic representation can take the means of processing times in the given distribution. Then, the original stochastic problem equals the optimal schedule solution in the offline model.

2.4 Workflow Manager System

The scientific community is increasing the use of workflows to organize their computing experiments. As a consequence, workflow manager systems demand middleware that executes applications on computing resources. In relation to scheduling processes, Workflow Manager Systems (WMS) present three main characteristics: specification, composition and information retrieval.

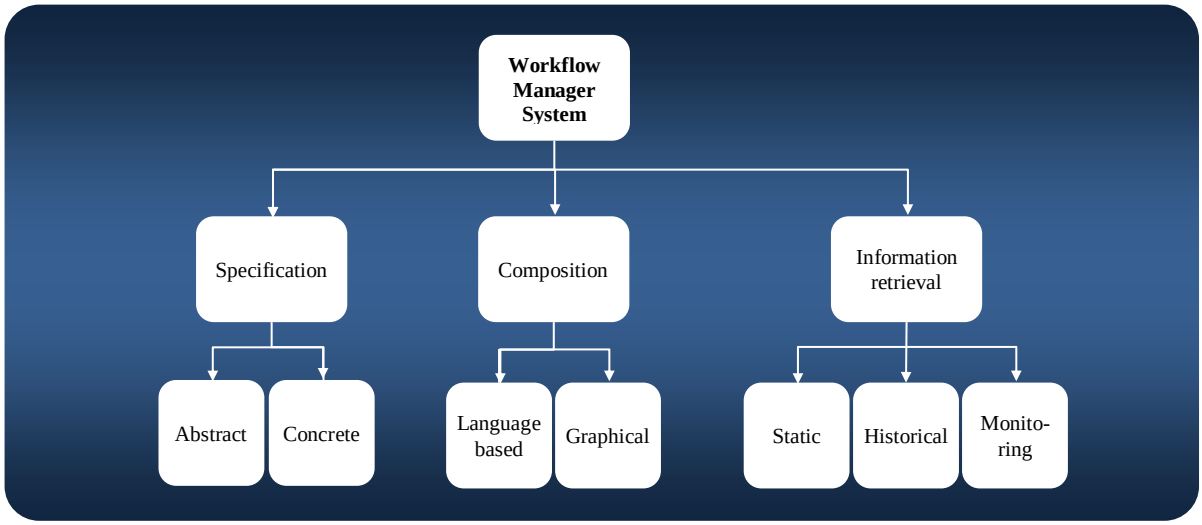


Figure 8. Taxonomy of Workflow Manager System.

Workflow Model Specification

In an abstract description, a task description does not specify a particular computer machine for its execution. Abstract models enable application owners to describe workflows without concern for low level implementation details. Tasks in an abstract workflow model are portable to other computational systems. In contrast, concrete workflow descriptions fix tasks to specific resources.

Workflow Composition System

In relation to language-expressed workflows, applications are literally described using language syntax as XML (Extensive Markup Language). Graph-expressed workflows enable application owners to express applications in graphical form. Graph-expressed models require low level details; hence users focus on a higher abstraction of the application. Unified Modeling Language (UML) is a well know graphic modeling system for workflows.

Information Retrieval

WMS retrieve information in three different ways: static, historical and dynamic. Information that does not change over time is referred to as static. This information includes VM parameters as number of processors, operating system, memory size, and VM identification number. Historical information is retrieved from past operations. Workflow

managers utilize historical data to predict future behavior of resources. Monitoring task execution is included as a dynamic retrieval of information. The status of task execution and resource behavior information is used to re-schedule tasks after failure.

2.5 Cloud Computing System

In this study, the cloud computing taxonomy will focus on the technical elements of virtualization and system architecture.

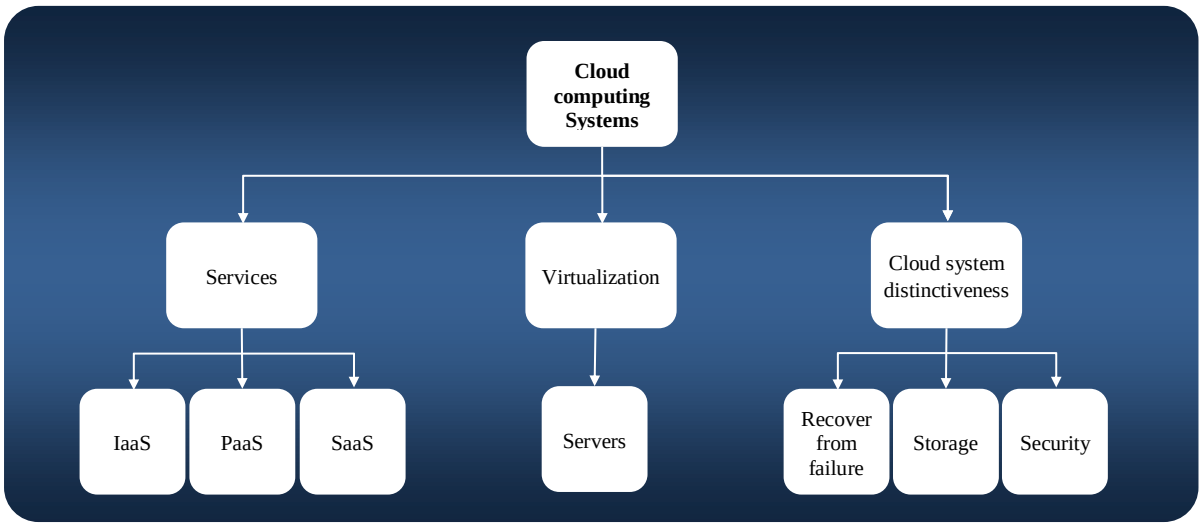


Figure 9. Taxonomy of Cloud computing systems.

Services

Cloud computing provides three main services:

- *Software as a Service (SaaS)*, a tenant platform usually referred to as an application service. On it, cloud owners develop and offer a broad range of applications ready for use. Example of such applications are Google Apps, Salesforce, Workday, Concur, Citrix GoToMeeting and Cisco WebEx
- *Platform as a Service (PaaS)*, a platform with all the components required to develop, test and host web based applications, such as Microsoft Azure
- *Infrastructure as a Service (IaaS)* delivers a service in the form of virtualization. The most important feature of this service is its payment scheme. Clients are able to scale up and down resources and reimburse only the deployed resources

Virtualization

Virtualization refers to the abstraction of hardware and operating systems in computing systems [24, 25]. Server virtualization, a key concept in cloud environments, maps physical resources to multiple partitions. These partitions can be dynamically created, expanded and moved in order to satisfy computational demands.

Cloud System Main Distinctiveness

Cloud resources are not free from failure. In case of these occurrences, the cloud administrator is ready with an instant backup to recover from failure avoiding any disruption in the application's operations. Security, in terms of data, is an important concern for specific cloud customers. For instance, corporate clients often manage private data held on behalf of customers and employees that must be kept protected requiring specific security privacy policies. Users are able to use the cloud resources as a storage system without worrying about where the actual files are saved. Even so, the main issue with storage is reliability, if a cloud provider is not able to provide a security warranty, a high number of clients will avoid their services.

2.6 Summary

This section provided taxonomy on classifying fundamental characteristics of cloud environments to analyze/solve the workflow scheduling problem. The importance of this taxonomy lies in its great influence on the decision making process of building specialized scheduling that plays a key role in the execution of scientific applications on cloud environments.

3 Literature Review

3.1 Preliminaries

This chapter describes the existing related work and concludes with the open issues not considered by current schedulers. Firstly, Section 3.2 presents the existing solutions to the scheduling problem. Secondly, section 3.3 exhibits how current solutions have left open important issues.

3.2 Scheduling Algorithms

This section presents a survey of scheduling algorithms. Based on their performance, algorithms are divided into four categories: (1) Basic scheduling techniques, (2) Random guided procedures, (3) Critical path based scheduling frameworks and (4) Iterative and/or evolutionary searching algorithms.

3.2.1 Basic Scheduling Techniques

Based on the taxonomy previously presented, basic scheduling techniques have a centralized architecture employing time as their only optimization objective. These

schedulers have a dynamic mechanism to allocate tasks; as a consequence, they are suitable for solving online scheduling problems. A summary of the main characteristics is presented at the end of this subsection.

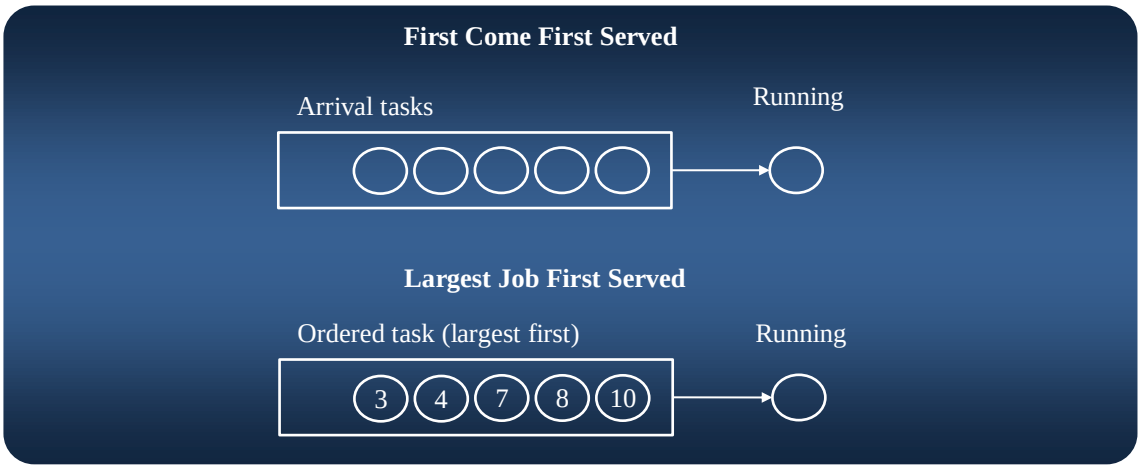


Figure 10. Basic scheduling techniques examples.

Adaptive First Come First Served (*AFCFS)*

The AFCFS is a *First Come First Served* modified technique [6, 26]. The AFCFS submits jobs to computing resources as they become available. If more than one computing resource is available for a job, the AFCFS assigns the job to the resource that contributes the lowest execution time. AFCFS gives priority to jobs requiring low processing time. Due to this policy, AFCFS results in an increased response time for larger jobs.

Largest Job First Served (LJFS)

The LJFS assigns priority to each job based on its computing demands [27]. LJFS gives the highest priority to jobs requiring the greatest computing processing time. To execute an application, LJFS firstly prioritizes every job, and then it sends them for execution. This technique provides a fast response to highly parallel jobs.

Myopic

Myopic is considered to be one of the simplest scheduling procedures . It focuses on a single task at a time and assigns tasks to computing resources in an arbitrary order. The scheduling process continues until all tasks are distributed. A myopic algorithm has been implemented in workflow manager systems such as Condor [28].

Min–min

Min-min [29, 30] is a scheduling heuristic that assigns priority to jobs based on their expected completion time (ECT). Min-min groups tasks and then assigns them to a resource that contributes the minimum ECT. The process moves to the next group of tasks until all workflow tasks from all groups are assigned to a computing resource. In contrast to Myopic, Min-min considers a group of tasks during its decision making, whereas Myopic considers one task at a time. Min-min has been used in grid projects, such as Pegasus [31].

Max–min

The Max-min has a similar approach to Min-min. The main difference is that Max-min assigns the highest priority to tasks requiring the longest execution time [29, 30]. In every Max-min iteration, tasks with the maximum estimated processing time obtain the resources that complete them within the earliest time. Max-min's objective is to minimize total execution time by executing the longest tasks on the fastest computing resource. Max-min has been used in Pegasus [31].

Summary

The aforementioned techniques are schedulers based on a limited set of rules. Their main characteristics are their irrelevant overhead scheduling time and their limited efficiency. As a consequence, their application to schedule workflow applications to computer resources is inefficient. They do not consider task interdependencies, a basic characteristic in every scientific workflow. In contrast, this study considers task interdependencies as well as file dependencies among computational tasks.

3.2.2 Random Guided Algorithms

In order to allow schedulers to consider task interdependencies, different studies developed algorithms that exploit randomness in the search for a scheduling solution. These solutions have a centralized scheduling architecture and optimizing objectives such as execution time. Their main difference from previous solutions is that random guided algorithms produce a static scheduling prior to execution and attack the scheduling problem as an offline problem type. A summary of the principal characteristics is presented at the end of this subsection.

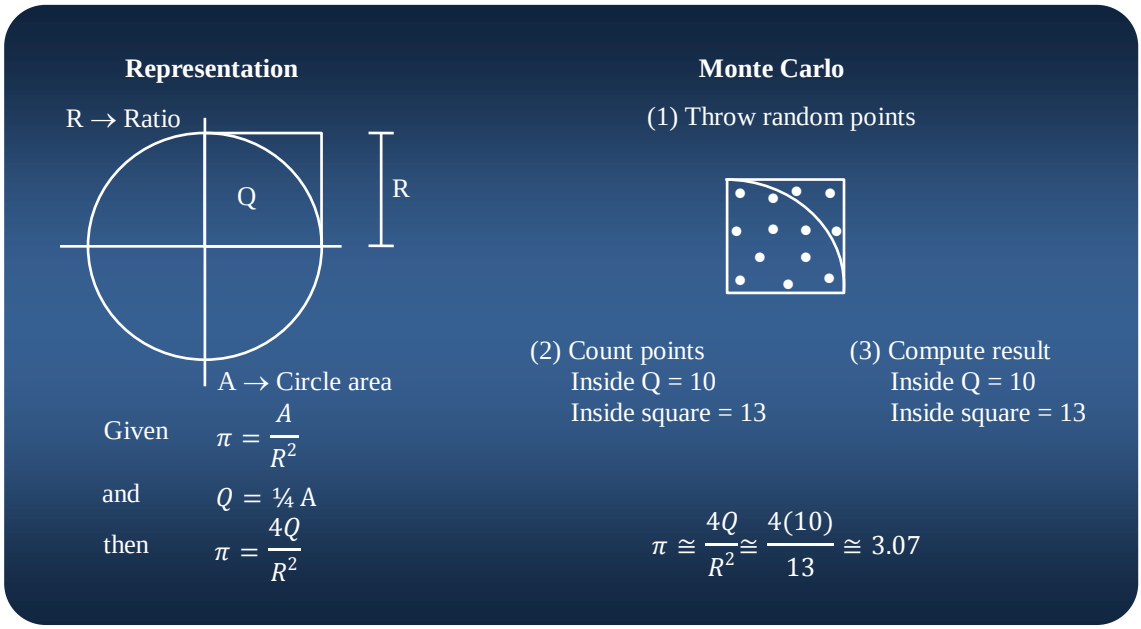


Figure 11. Monte Carlo, an example of an algorithm based on randomness in finding the value of π.

Monte Carlo Algorithm

The Monte Carlo [32] (Figure 11) is an algorithm based on randomness, it is not an exact method but its process requires a low computation power [33]. Monte Carlo builds solutions based on randomness and probability distribution. It has four main stages: it firstly defines a solution space, then it assigns random values to its inputs, after that it evaluates the solutions produced and finally it aggregates the results. The Monte Carlo approach is employed to solve the scheduling problem considering tasks as inputs; tasks aim to obtain a resource for their execution. Applications that apply the Monte Carlo algorithm to scheduling problems can be found in [34, 35] .

Greedy Randomized Adaptive Search Procedure (GRASP)

GRASP [36, 37] conducts a greedy search for an optimal (or close to optimal) scheduling solution. GRASP, a randomized optimization technique, generates a group of solutions at each step; it keeps a record of the best solution, and at the end of the entire process, the best solution is presented as the final solution.

Non-Evolutionary Random Scheduling

The Random Scheduling (RS) algorithm presented in [38] matches and schedules interdependent tasks to computing resources. RS firstly organizes tasks into a randomized order while maintaining task precedent constraints. Then it assigns these tasks to the available computing resources. RS presents similarities to evolutionary techniques such as GA but it sacrifices output efficiency for lower memory usage, less algorithm complexity as well as requiring fewer parameters to be set up.

Summary

As their name states, random guided algorithms exploit randomness to hopefully obtain a satisfactory solution. The main disadvantage of these algorithms is scalability. Due to randomness, the time to produce solutions becomes unacceptable as the size of a workflow increases. In contrast, the scheduling solutions offered in this study reduce randomness by guiding the search for solutions. Additionally, the solutions exploit parallelism, data replication and system optimization.

3.2.3 Critical Path Based Scheduling Algorithms

With the aim to produce an efficient scheduler in terms of scheduling overhead time, a great number of schedulers use the critical path at a preliminary scheduling stage. These schedulers have a central architecture to produce their decision making. All of them optimize at least execution time while some others allow including a particular constraint. Critical path based schedulers produce a static configuration for offline and stochastic scheduling problem types. A summary with the principal features is presented at the end of this subsection.

Heterogeneous Earliest-Finish-Time (HEFT)

HEFT is the pioneer in using the critical path for scheduling [13]. HEFT focuses on the scheduling of applications targeting a minimal execution time and a low scheduling overhead time. HEFT firstly identifies the critical path. Then, it selects the first job on the critical path and assigns it to the computing resource that has the potential to execute it with the earliest finishing time. The algorithm continues until all jobs are scheduled.

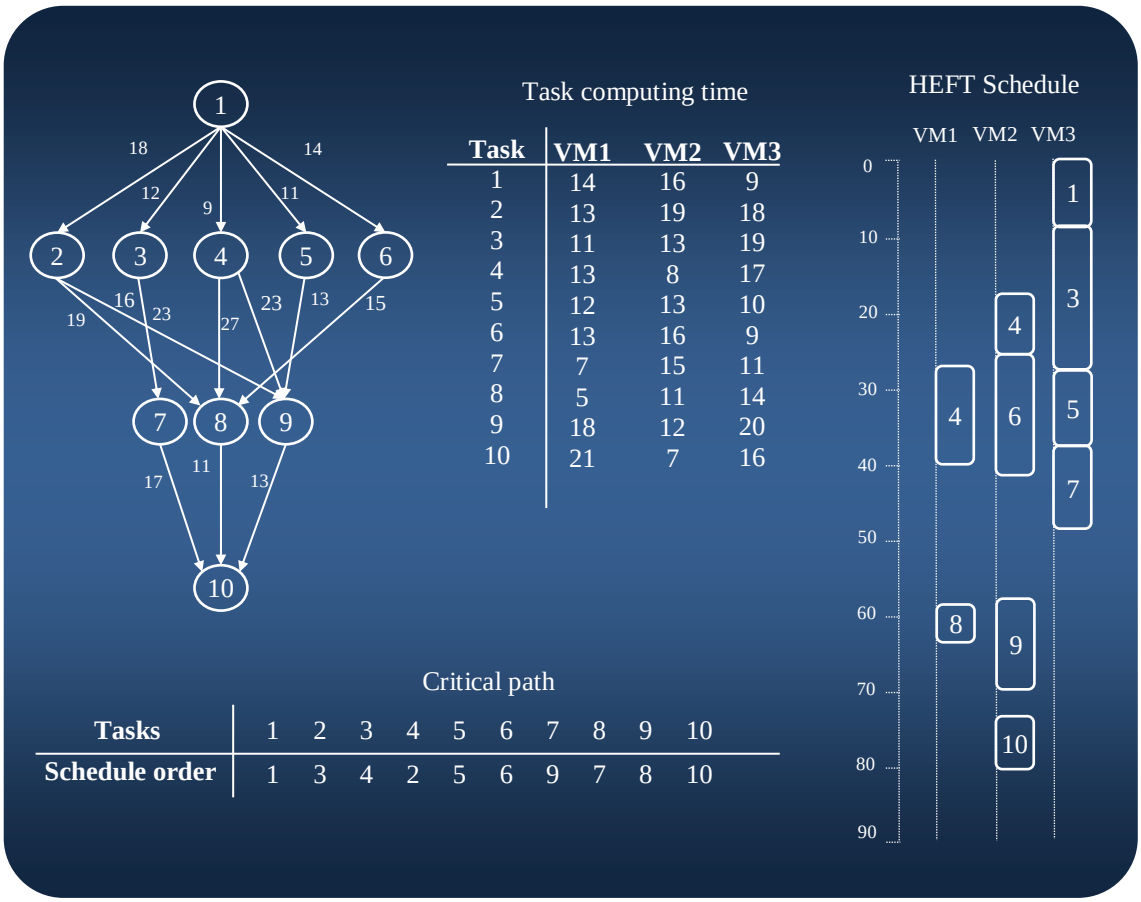


Figure 12. HEFT, a distinguish scheduler based on the critical path.

Critical Path and Area Based (CPA) scheduling

The CPA (Critical Path and Area based scheduling) incorporates the concept of area based scheduling to minimize the total workflow execution time [10, 11]. CPA considers the area scheduling as the product of quantity of processors and time to execute a given application. CPA first allocates one processor to each task from a preliminary group of

processors. Then, on every repetition, CPA allocates an additional processor to the most critical task i.e. the task in the critical path which would contribute the greatest benefit. The algorithm stops when the execution time of the critical path goes below the average execution time between all processors.

Critical Path First (CPF)

CPF is a scheduling algorithm targeting resource performance optimization [14, 39]. CPF "stretches out" the scheduling efficiently and keeps the critical path length to a minimum in order to optimize system performance. CPF first assigns critical path tasks to resources to achieve a minimum execution time and then it assigns the rest of the tasks to resources in a way that has minimal impact on their execution time. The main objective of this approach is to exploit resource availability and maintain total execution time.

Heterogeneous Duplication-Based Scheduling (HDBS)

HDBS [12] incorporates the task duplication concept and the critical path to optimize execution time. The HDBS algorithm initially labels critical path tasks as CP, tasks with direct relation to any CP as in-branch (IB) tasks and the rest are labeled as out-branch (OB) tasks. The HDBS starts the scheduling list with the initial CP task, then it starts adding subsequent IB and CP tasks to the list while preserving precedence constraints. Then, every unscheduled descended task of CP is recursively duplicated and integrated into the scheduling list in order to minimize the execution time of the critical path. Finally, OB tasks are added to the scheduling list. Algorithms duplicate OB tasks whenever they increment the finishing time.

Dynamic Critical Path (DCP)

The DCP [40] is an algorithm based on the principle of shortening the longest path by moving jobs to an earlier stage in the execution plan. The original approach was designed to distribute tasks to a group of identical resources. To attack the scheduling problem for heterogeneous resources, the algorithm:

1. Puts all tasks into a queue on a single resource and leaves the rest of the resources empty.

2. The DCP employs a term called earliest/latest starting time (AEST/ALST) to represent the possible earliest/latest time to start execution of a given task on a particular machine. Whenever a given task has a smaller AEST on a different machine the algorithm moves the task. Nevertheless, the scenarios with different types of resource mean that runtime of a task may be different over different resources. As a consequence, authors create another term named absolute earliest/latest finish time (AEFT/ALFT) referring to the possible earliest/latest finish time of subtasks on its current machine.

3. The DCP approach finishes if all the tasks have been scheduled once. Yet, the authors discovered that for applications with parameter-sweep tasks, the algorithm will be able to increment effectiveness by 10%-20% whenever the algorithm is run again on the scheduled result. Nonetheless, the time spent to schedule must also be considered.

4. The algorithm takes into consideration data transfer time, even though the algorithm does not consider inter-processes between subtasks. As a consequence, the algorithm simply removes the terms that are related to transfer cost.

5. Whenever a subtask is scheduled, the algorithm checks whether dependent subtasks are assigned to the same resource, in those scenarios a correct order of execution is given to every job in the subtasks. Results exhibited that the DCP effectively reduces makespan by 30% for applications that exhibit sweep tasks.

Summary

Critical path schedulers achieve efficient results in terms of scheduling overhead time. Additionally, they deliver solutions with satisfactory execution times, even though the inherent scheme of the critical path dismisses key configurations that have the potential to reduce execution time. For instance, it uses all the available computer resources without considering a reduced pool of resources that can lead to an optimal scheduling configuration. To overcome these issues, this study includes important scheduling techniques as load balancing and data replication in order to evaluate different configuration that have great

potential to become an optimal solution. Additionally, it considers the monetary cost on its model, as a consequence, the solution presented here delivers a trade-off between execution time and economical cost.

3.2.4 Iterative Guided Searching Algorithms

Recent studies delivered solutions based on different optimization theories to offer superior results as compared with other solutions. Iterative algorithms, as these schedulers will be referred to, consider a centralized architecture that optimize selected objectives. The main purpose of employing these theories is to produce a static scheduling considering an offline and/or stochastic problem type. The main difference with the rest of the schedulers is that iterative algorithms do not build a single final solution. Instead, these algorithms endorse an iteration of a population of solutions in order to converge to an optimal (or close to optimal) solution. Most of these iterative searching algorithms are based on firm theories in economy, biology, and chemistry.

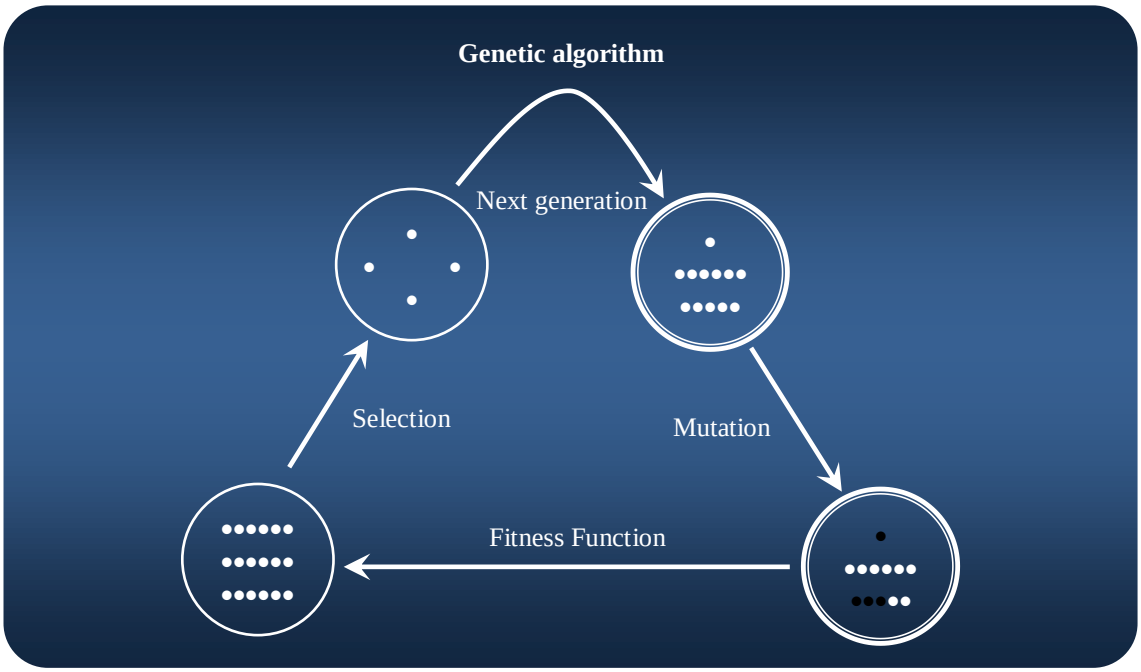


Figure 13. Genetic algorithm

Genetic algorithm

The Genetic Algorithm (GA) is a metaheuristic inspired by genetic evolution. GA is a robust technique with a great capability to discern an optimal solution from a search space [41, 42]. The GA consists of a population of chromosomes that evolve to obtain the strongest elements. Selection, the first GA phase, starts every new generation by selecting chromosomes from a previous generation. Then, at the crossover operator phase, the GA splits and mixes pairs of chromosomes to give birth to offspring. The mutation operator phase literally mutates chromosomes, altering their natural composition to incorporate new variation to the population. Finally, at the elitism operation stage, the chromosomes with the highest fitness values are copied to the next generation. GA terminates when it meets a finishing criteria, usually at a specific number of iterations or when it gets a minimum fitness value. Different research studies have adapted the GA to the scheduling problem [41-50]. In these adaptations, chromosomes represent scheduling solutions and fitness function measures parameters such as runtime, security or monetary cost. The success of every research contribution relies on the approach to model the problem and the manner to adapt the GA to the scheduling problem. Even so, most genetic based scheduling algorithms fail to adapt the methodology without the excessive randomness from the original algorithm, as a consequence they require excessive time to converge on a final solution. To cope with this issue, this study introduces an adaptation of the GA to solve the scheduling problem. The proposed algorithm modifies the chromosome operator to combine parts of chromosomes that already have proved optimal results. Also, its modified mutation operator includes and dismisses computer resources during the scheduling process allowing the modified algorithm to converge to the specific number of resources that application requires. Additionally, this study succeed in designing an algorithm with a reduced complexity.

Particle Swarm Optimization (PSO)

PSO, inspired by bird swarming, has similarities with Darwinian theories as a genetic algorithm [51, 52]. PSO is used as an optimization process to find a solution for nonlinear problems. It encompasses a population of particles swarming to reach the position that maximizes (or minimizes) its evaluation function value. In the initial population, particles acquire random positions and velocities. Then, the process combines particles with their best position registered and with the best particle, particles chase the elements with the highest evaluation function values. PSO terminates when it meets a finishing criteria such as number

of iterations. In order to implement the PSO with regard to the scheduling problem, different research studies utilized the discrete PSO version [53-60]. On these solutions, particles comprise a complete scheduling algorithm where position represents tasks assigned to computational resources and velocity dictates a probability to move them to a different resource. One common issue with PSO is that in practical experience results are not presented constantly, i.e. the method performs better at some times than others. To cope with this issue, this study presents an adapted PSO scheduler enhanced with a super particle that collects the most popular particles' elements instead of only selecting a global best particle. By doing so, the algorithm is able to provide repetitiveness and a warranty of not getting trapped in a local optimum. Additionally, this study produces a novel scheduling reconstruction from particles' velocities that allows the adapted technique to schedule groups of consecutive tasks to the same resource.

*Chemical Reaction Optimization (*CRO*)*

CRO emulates chemical reactions where molecules interact with each other with the objective of obtaining the minimum state of free energy. Chemical reactions frequently occur in closed containers with a fixed number of molecules. Collisions during the CRO process attempt to modify the molecule's structure. Whenever a molecule hits another element or container wall, energy is released. If the energy reaches a predefined value, it alters the molecule(s)' internal structure. The four collision types in the CRO are: on-wall ineffective collision, decomposition, inter-molecular ineffective collision, and synthesis. On-wall ineffective and decomposition collisions consider only one molecule. Inter-molecular ineffective and synthesis collisions contemplate a pair of molecules. Inter-molecular collision yields a pair of new molecules, while synthesis fuses a pair of molecules into one. On-wall ineffective and decomposition collisions produce results close to the molecular to molecular collisions.

At each CRO iteration, a collision is selected. Firstly a molecule energy value is set, then a temporary value is randomly selected between the interval [0,1]. If the value is greater, it will cause a unimolecular collision. If it is less, then intermolecular collisions occur. The algorithm uses α and β as thresholds for the criteria of unimolecular and intermolecular collisions selections, respectively. α refers to the maximum number of collisions that can take place. In the cases that a molecule hits a number larger than α, then a decomposition takes place. β refers to the least amount of energy a molecule should have. For a couple of

molecules m1 and m2, the synthesis is activated when the energy m1 < β and m2 ≤ β. Whenever this sentence is false, then an intermolecular ineffective collision occurs.

CRO finishes when a stopping criterion is reached, then the best solution is reported as the final solution. Different studies adapted the CRO to the scheduling process. Adaptations such as [61, 62] resulted in the molecules representing a complete scheduling solution and collisions offering the possibility of molecules to interchange their configuration. Due to the nature of the CRO, it emulates a hermetic container without the possibility of including an external factor such as a new computer resource. This limitation forces the algorithm to consider a limited number of configurations that may not end in an optimal solution. In contrast, this study proposes a genetic algorithm with a mutation operator that specifically injects new resources to the process; in this way, the algorithm has greater possibilities to converge to an optimal solution on a reduced number of iterations.

Immune System

The immune system is a biological self-defense mechanism in organisms to protect themselves from bacteria and viruses. The immune system defense has two main phases: innate and adaptive systems. During a virus or bacteria attack, organisms uses the innate system as a first line of defense, it uniformly fights aggressors right after their attack with biochemical substances and blood cells. If intruders prevail over the innate defense, then the adaptive system comes into play. The adaptive system mainly uses B and T lymphocytes from white blood cells to prevail with cellular immunity creating antibodies to dissolve intruders. Important research studies successfully adapted the immune system to the scheduling problem as in [63-65]. In this study authors designed a scheduling algorithm where tasks aim to survive a virus attack. The success of the adaptation of the immune system to the scheduling problem relies on the representation of its actors, for instance, B and T lymphocytes can be represented by the fastest computer. In these scenarios the optimization of monetary cost would struggle as the algorithm is already making a decision before the start of the scheduling process. In contrast, this study proposes scheduling algorithms that start the scheduling process with a variety of solutions and subsequently allows them to include/dismiss solutions in order to increment the performance of its solutions.

Auction Theory

In economy theory, auction is an instrument to sell assets to a group of bidders. The bidding mechanism, the core of auctions, consists of bidders (consumers) offering a price for a product or service to providers. In English auctions, consumers raise their offer until the highest bid is reached. In Dutch Auctions, product prices decrease until a consumer decides to purchase the asset. In Sealed Auctions, potential consumers secretly submit and offer their bid, the winning bidder is the one with the highest bid. The Vickrey Second-Price Auction is a modified sealed auction where the second best offer is the winner of the auction. In Continuous Double Auctions, bidders and providers submit their proposals to each other. The process continues until a match is made. These auction theories have the capabilities to solve the scheduling problem. Examples of these theories applied to the scheduling problem appeared in [66, 67]. On them, bidders are users aiming to obtain computational resources to execute their application at the best price. Even so, auction theory has practical disadvantages. For instance, in the Vickrey auction [68], bidders need to offer a bid to each good they would like to buy. When applied to the scheduling problem, it is impossible to run the auction as theory dictates due to the high number of combinations that tasks and computer resources produce. The nature of the scheduling problem requires a solution that bypasses the evaluation of every single option and yet produces outstanding solutions. To cope with this issue, this study proposes solutions that directly exploit parallelism, balance loads across resources and discover key distribution of tasks among resources. These specific rules prevent solutions from analyzing configurations that have poor performance.

3.3 Open Issues

The related work described in Section 3.2 presented a review of the state of the art of scheduling algorithms. The exhibited studies have delivered different contributions to existing issues such as: straightforward decision mechanisms [6, 28-31], random allocation of tasks [33-36, 38] , optimization of execution times based on pre-organized lists [10-14] and outstanding scheduling frameworks based on different optimization theories [41-47, 51-57, 61-63, 66-68]. The open questions this study targets follow specific characteristics drawn from the taxonomy previously presented. Figure 14 presents a selection of the taxonomies to execute scientific workflow applications.

Firstly, the efficient execution of scientific workflows requires a *centralized* scheduling engine to fully manage workflows during execution as developed in [6, 10-14, 28-31, 33-36, 38, 41-47, 53, 55-57, 61-63, 66, 67, 69]. In addition, a *static* analysis is required prior to execution in order to consider every scheduling possible configuration as proved in [10-14, 33-36, 38, 41-47, 53, 55-57, 61-63, 66, 67]. Since the full workflow is known prior to execution, the scientific problem considers an *offline* and/or *stochastic* model as adopted in [10-14, 33-36, 38, 41-47, 53, 55-57, 61-63, 66, 67]. Equally important scientific applications exhibit dependencies among their tasks, for this reason scheduling engines must accept *DAG* style workflows similar to [41-47, 53, 55-57, 61-63, 66, 67]. Concerning workflow management systems which are an important part of the experimentation process, WMS are required to accept concrete description configurations from the scheduler as per Condor [28] and Pegasus [31] frameworks. In relation to the cloud system, the execution of the application requires *virtualization* of computing resources in the form of Virtual Machines. For instance, Amazon EC2 is a well-known public cloud provider with *IaaS* availability.

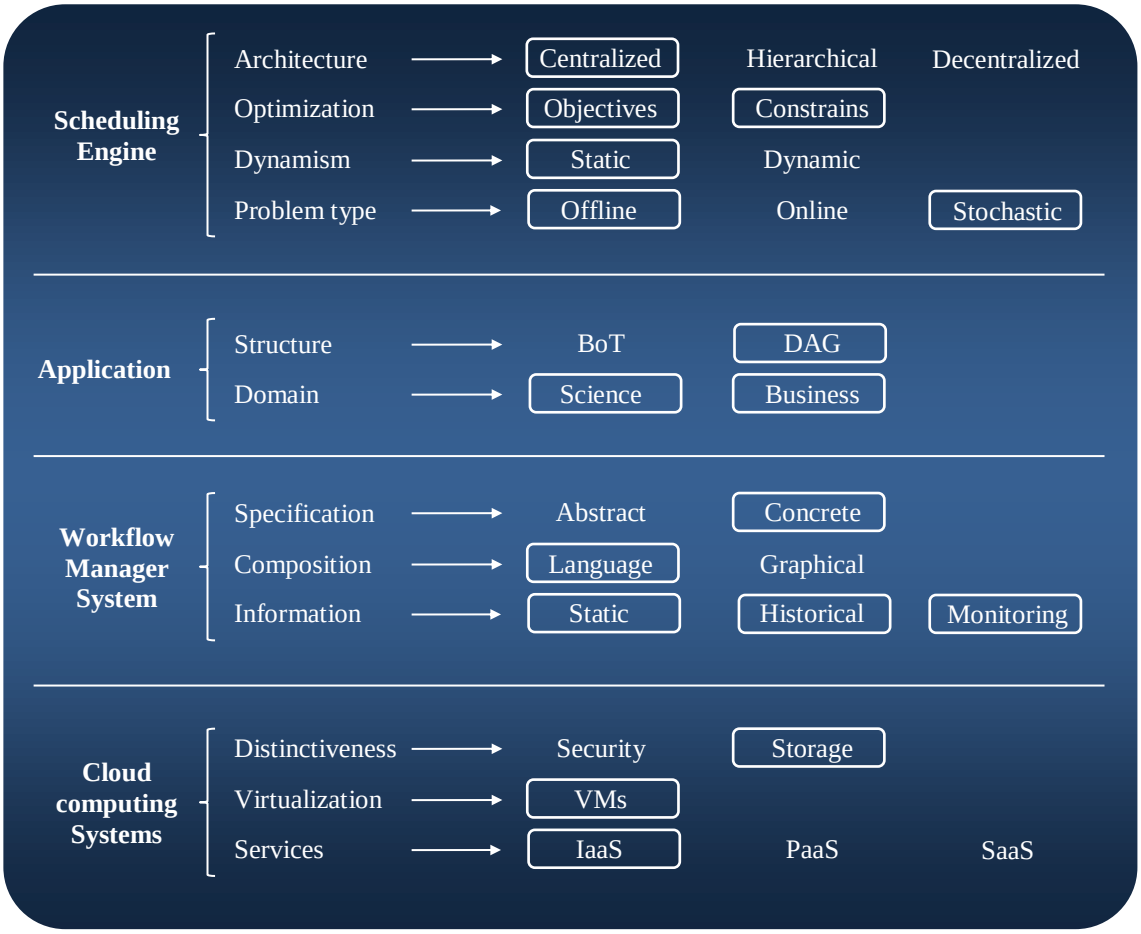


Figure 14. Taxonomy selection for an environment to execute scientific workflows.

3.4 Summary

Subsection 3.2.4 presented exceptional scheduling solutions based on diverse optimization theories; solutions exhibited outstanding results compared with the algorithms presented in Sections 3.2.1 - 3.2.3 . This study recognizes that authors of the mentioned algorithms maximize parallelism, exploit data replication and duplications of jobs, and challenged both computing and data intensive applications. Even so, these scheduling techniques made modest contributions to: define the number of VMs, increment system utilization, provide users with different scheduling plans, contemplate a cost model based on public cloud charging scheme and analyze the optimization of conflicting objectives such as time and cost. Any attempt to produce superior results must involve a specific modification of their decision-making engine. The open issues this thesis addresses are the efficient execution of scientific workflows on cloud environments minimizing makespan and monetary cost as well as defining the number of required VMs. To accomplish this goal, this

thesis proposes scheduling solutions that not only exploit parallelism but search for efficient alternatives and adapt important optimization theories such as genetic algorithm and particle swarm optimization.

4 BaRRS: A solution to the scheduling problem

4.1 Preliminaries

This chapter presents a triple mechanism to solve the scheduling problem. This apparatus is referred to as the "Balanced with Data Reuse and Replication Scheduler", BaRRS. Firstly, Section 4.2 presents the framework required to present BaRRS. Section 4.3 presents the problem statement, then Sction 4.4 describes in full detail the BaRRS approach. Section 4.5 describes the experimental setup to test BaRRS. Next, Section 4.6 presents the obtained results. Section 4.7 provides a discussion of the obtained results focusing on the optimization of objectives. Finally, Section 4.8 gives a conclusion with the most remarkable features of the BaRRS performance.

The main contributions of BaRRS are: (1) concurrently optimizing two important, yet conflicting objectives, i.e. runtime and monetary cost, (2) introducing a triple-mechanism scheduler based on load balancing, file reuse and data replication, and (3) exploring the trade-off between the aforementioned objectives through scheduling sample configurations. The core of this chapter has been published in the *Future Generation Computer Systems* journal [1].

4.2 Framework

The environment model consists of a set $\mathbf{VM} = [\mathrm{vm}_1, \ldots, \mathrm{vm}_\mathrm{v}]$ with v VMs. The characteristics of each VM are network bandwidth (vm_j^{bw}), number of cores (vm_j^{cores}), memory (vm_j^{mem}) and disk size (vm_j^{disk}). The model adopts hours as the minimum time unit to hire a computer resource, i.e. a VM. The VM hourly cost is given by $\langle c_1, \ldots, c_v \rangle$.

Every workflow $\mathrm{W} = [\mathrm{t}_1, \ldots, \mathrm{t}_\mathrm{n}]$ has n number of tasks. Tasks have a set of input files with size in_i^{size}. Parent tasks of t_i are given by $t_i^{parents}$. Before a task is executed, a central manager needs to transfer its respective input files to the respective VM. The time to transfer in_i^{size} is set by $\hat{t}_i^{transfer}$ while the task execution is given by $\hat{t}_i^{exe}$. Total estimated time to execute the *i-th* task is denoted as:

$$\hat{t}_i^{total} = \hat{t}_i^{transfer} + \hat{t}_i^{exe} \qquad \text{Eq. 1}$$

Figure 15 exhibits five scientific workflow examples extracted from [4]; each with a specific dependency pattern between tasks. A workflow level is a group of tasks with a single parent group. The parallelism, P, of a level is the number of tasks building the given level; for example, the Montage workflow in Figure 15.b has nine levels, where the second level has the maximum parallelism, *max(P)*, with six tasks. On each workflow, nodes are represented by a circle containing a single task t_i with its input set of file(s) of size f_i^{size}. Depending on how nodes are related, five main workflow structures/distributions are highlighted: (1) *Pipeline* structure connects nodes serially, (2) *Data distribution* highlights a set of nodes requiring a single set of input files, (3) *Data aggregation* represents nodes requiring files from at least two other nodes, and (4) *Data redistribution* highlights nodes combining structures (2) and (3) requiring and producing files for multiple nodes. In order to organize workflow analysis, we define *w-level* as the number of workflow levels and *parallel-tasks* as the maximum number of tasks a workflow can execute in parallel. For example, the Montage workflow has nine *w-levels* and *parallel-tasks* has a value of six as exhibited in Figure 15a.

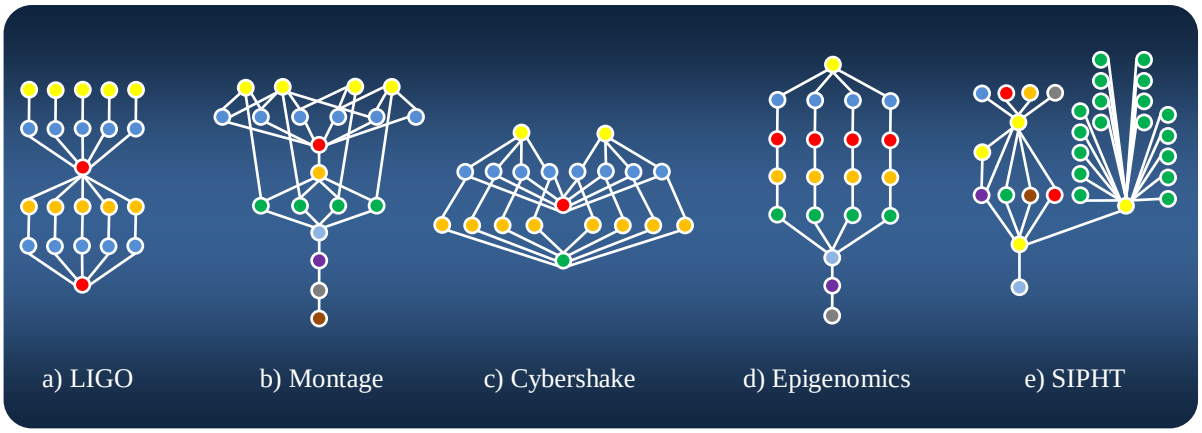


Figure 15. Example scientific applications.

This model considers the following assumptions: (1) the system executes *W* at a time and contemplates computing and data-intensive workflows, (2) each VM works under a particular bandwidth vm_j^{bw} that is assumed to be fixed during the execution of *W*, and (3) resources are requested from a cloud provider prior to execution and released after the execution of each task in the workflow. Additionally, it is assumed that users will provide the estimated execution time $\hat{t}_i^{exe}$ for all tasks in *W*. The file transfer time is given by the total size of file(s) divided by the minimum bandwidth value between VMs, expressed as:

$$\hat{t}_j^{transfer} = \frac{in_i^{size}}{min\left(vm_p^{bw}, vm_i^{bw}\right)} \qquad \text{Eq. 2}$$

Where vm_p^{bw} and vm_i^{bw} refer to the VMs executing t_i and its parent t_i^{parent} respectively.

4.3 Problem statement

The scientific workflow scheduler formulates the scheduling problems as a weighted optimization problem of two objectives (monetary cost and runtime). It assigns tasks to VMs to minimize execution time and monetary cost based on user requirements. To formulate this problem, the considered tasks are distributed among the VM's queues, $\left[vm_j^{queue}, \ldots vm_v^{queue}\right]$. A queue is defined as a decomposition of a set into disjointed subsets whose union is the original set. Based on this model, the scheduling problem is defined as finding the corresponding elements of each VM queue to maximize the following augmented

objective function (*F*):

$$F = w_1 \frac{(max^{time} - Runtime)}{(max^{time} - min^{time})} + w_2 \frac{(max^{cost} - Cost)}{(max^{cost} - min^{cost})} \quad \text{Eq. 3}$$

Variables max^{time}, min^{time}, max^{cost} *and* min^{cost} are continuously updated during the scheduling process reflecting the best and worst VM configurations. These values represent the maximum and minium values of runtime and monetary cost.

Runtime is defined as the maximum time taken by the slowest or least powerful VM to execute the current queues of jobs, expressed as:

$$Runtime = Max_{j=1}^{|VM|}[vm_j^{time}] \quad \text{Eq. 4}$$

Where, $vm_j^{time}$is the time to execute vm_j^{queue}on vm_j

$$vm_j^{time} = \sum_{i=1}^{|vm_j^{queue}|} t_i \quad \text{Eq. 5}$$

Cost or *Monetary* is defined as the sum of runtimes of each VM multiplied by its respective cost, expressed as:

$$Cost = \sum_{j=1}^{|VM|} \lceil Runtime \rceil vm_j^{cost} \quad \text{Eq. 6}$$

Since application owners do not have tools to accurately estimate total execution time or monetary cost for their workflows, our approach offers an abstract and flexible way to choose a particular scheduling configuration driven by w_1 and w_2 representing Execution time and monetary cost optimization weights, where:

$$w_1 + w_2 = 1 \quad \text{Eq. 7}$$

In this way, the workflow owner gives a percentage weight to constraints based on his needs.

4.4 BaRRS Approach

In order to achieve the aforementioned goals, BaRRS produces an estimation table for large workflows. This table is presented to application owners for comparing monetary costs and execution time tradeoffs for executing their large workflow considering heterogeneous VM configurations (e.g. CPU Type, CPU Speed, cores, memory, renting cost, etc.). Firstly, BaRRS estimates results for a subset of VMs denoted as **subVM** where $\boldsymbol{subVM} \subset \boldsymbol{VM}$ and $|\boldsymbol{subVM}| = \alpha|\boldsymbol{VM}|$ size $(0 < \alpha \leq 1)$. Next, BaRRS employs a trade-off analysis technique [44, 70, 71] for building the complete set of solutions considering an exhaustive set of **VM configurations**. In this chapter, the trade-off is modeled by the exponential shape graph, $f_e(x) = A_e \times \exp(xk_e)$, that maps the number of VMs to the execution time and its respective monetary cost. As in [44], this study found that exponential function has a lower mean square error (MSE) in comparison to the linear regression and other distributions. Hence an exponential graph was selected to model the scheduling estimations trade-offs. Given that $subVM_i^{time}$ is the execution time of *ith* configuration in $\boldsymbol{subVM}$ then $A_e = \max(subVM_i^{time})$ where $i = 1,2, \dots v$. From $f_e(x)$, each VM configuration $x_i = subVM_i$ produces a different k_i:

$$k_i = \frac{\ln\left(\frac{f_e(x)}{A_e}\right)}{x_i} \quad \text{Eq. 8}$$

Then,

$$k_e = \frac{\sum_{i=1}^{|subVM|} k_i}{|subVM|} \quad \text{Eq. 9}$$

Finally,

$$f_c(x) = \sum_{j=1}^{|VM|} \lceil f_e(x) \rceil vm_j^{cost} \quad \text{Eq. 10}$$

An example of $f_e(x)$ and $f_c(x)$ is shown in Figure 16a-b. Figure 16c presents an example of a complete trade-off graph with all possible combinations of VM configurations with their respective runtime and monetary cost.

This algorithm includes the parameter $0 < \alpha \leq 1$ to control the number of estimations in order to decrease the scheduling overhead time. For $\alpha = 1$, BaRRS behaves as a brute force algorithm because it produces all possible configurations with their respective runtime and monetary cost to execute a workflow. For $\alpha < 1$, the number of configurations is uniformly distributed among all possible configurations. As an example, for $|VM| = 10$ and $\alpha = 0.5$, BaRRS will produce five $(\alpha|VM|)$ estimation points; they will be for $x = \{2,4,6,8,10\}$.

4.4.1 Balanced with File Reuse and Replication Techniques Scheduling Algorithm (BaRRS)

Algorithm 1 presents the BaRRS heuristic. It first computes runtime and monetary cost for the selected number of estimations (line 1). Line 2 builds the complete set of solutions for the different possible number of VM resource configurations. Finally, line 3 presents the trade-off solutions to a user.

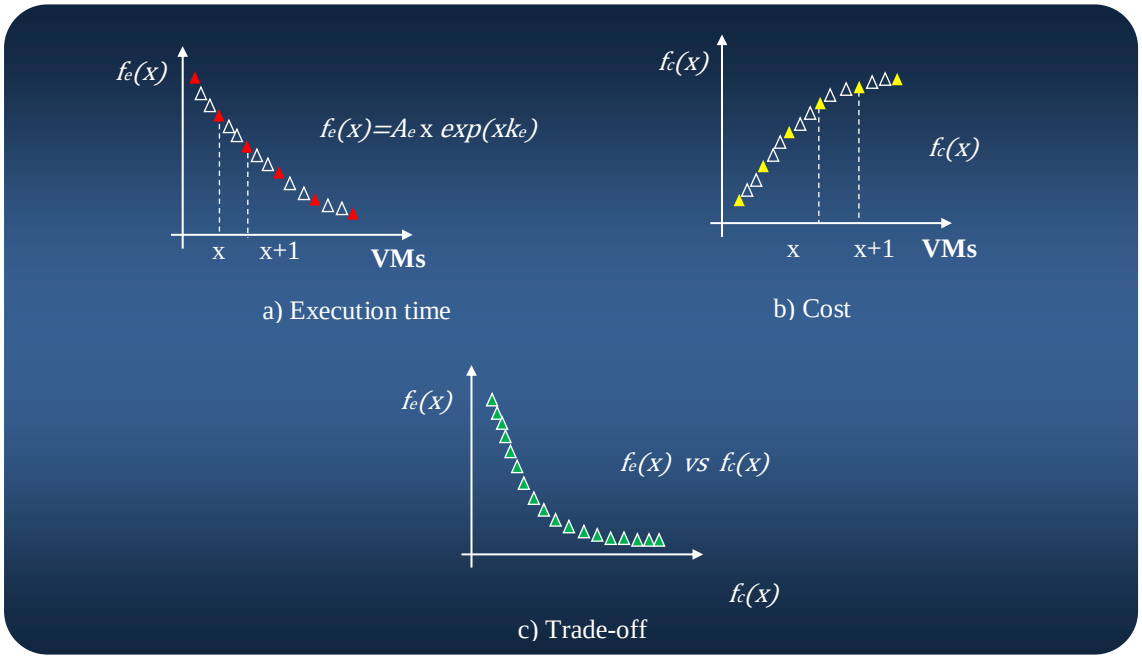


Figure 16. Trade-off frontier.

Algorithm 1 The BaRRS Approach

Input: Workflow *W*,
Output: Trade-off scheduling plans

1: **Estimate** Runtime, Monetary Cost (Algorithm 2)
2: **Build** Trade-off graph
3: **Present** solutions

In this context, the trade-off graph is defined as a set of solutions where each solution has a different number of VMs with its respective execution time and monetary cost. The set of trade-off values connected together are called the trade-off frontier. This trade-off follows a similar shape as the Pareto frontier [44, 71], the main difference is that the trade-off offers flexibility to analyze the complete spectrum of number of VMs, which is a key feature from this study.

The exponential function $f_e(x)$ is then used to obtain runtime where *x* is the number of VMs. Selection of k_e and A_e values requires significant attention since they drive the final trade-off shape [44]. In this work, the variable A_e corresponds to the maximum execution time for each approach. From the previous estimated solutions, k_e is obtained as

$$\log\left(\frac{f_e(VM)}{A_e}\right) = VMk_e \qquad \text{Eq. 11}$$

A different k is produced for each combination of VMs with a different MSE. Based on experimental practice, this study found the mean value of k_e offers minimum MSE. Figure 17 and Table 2 present an example of this concept. From an original graph $g(VM)$, five estimated *k* values produce different MSE, Table 2. Then $f(VM)$ reconstruct $g(VM)$ using $\bar{k}$ as shown in Figure 17. Since the left tail on the $f(VM)$graph does not match $g(VM)$ this approach does $f(1) = A_e$ to overcome this issue.

Table 2. Trade-off values and MSE examples

Parameter	Value	MSE
k_1	-0.2672	0.00043
k_2	-0.3683	0.00018
k_3	-0.3539	0.00014
k_4	-0.3068	0.00014
k_5	-0.2582	0.00056
$\bar{k}$	-0.3109	0.00011

$$\bar{k} = \frac{1}{n}\sum_{i=1}^{n} k_i \, .$$

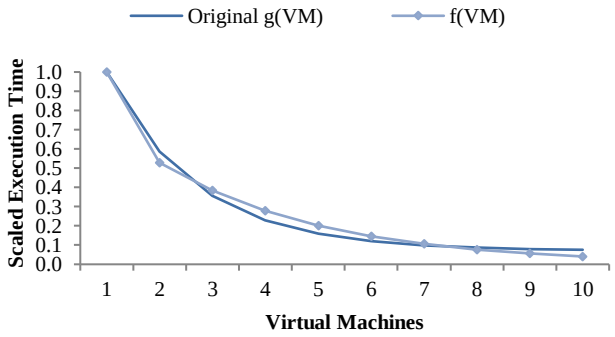


Figure 17. Trade-off values example.

4.4.2 Runtime and Monetary Cost Estimation

Algorithm 2 computes runtime and monetary cost. It first applies workflow contraction (line1). It then creates a VM pool. The size of the VM pool ($|VM|$) and the maximum estimated number are set in lines 2 and 3. The cycle from lines 4 to 7 obtains the best scheduling plan for the selected maximum number of estimations.

Algorithm 2 Runtime and Monetary Cost

Input: Workflow *W*, *VM* set
Output: Scheduling plan

1: Workflow contraction
2: $|VM| = max(P)$; *Solutions=Ø*
3: $MaxEstimations = \alpha|VM|$
4: **For** i = 1 to *MaxEstimations*
5: **Do** Scheduling Algorithm
6: *Solutions = Solutions + CurrentSchedule*
7: **End**
8: $BestSolution = Max_{i=1}^{total} F_i$

Algorithm 3 presents the Scheduling Algorithm. Its objective is to produce VM queues. This algorithm analyzes the total workflow task levels (|L|). It first enumerates the total number of descended tasks from the actual level (line 2). Then all the tasks are placed as in group *A* (line 3). Lines 4-11 add each task to the available scheduling queues based on file reutilization (line 6) and replication (line 7) techniques. Based on these techniques, the selected tasks are added to the scheduling queue that contributes the greatest Objective Function value.

Algorithm 3 Scheduling

Input: Workflow *W*, *VM* set
Output: Scheduling plan for the given *VM* set

```
1:    For l=1 to |L|
2:    s = total number of descend from actual level
3:        A ← [29]
4:        while parents ≠ ø
5:          for j=1 to s //for all descended
6:                File reuse and replication as per Algorithm
7:                Queue balancing as per Algorithm 5
8:               Add task(s) to the q with the highest
9:          End
10:       End
11:   End
```

4.4.3 File Reutilization and Replication

The file reutilization mechanism reduces the number of file transfers during workflow execution. This technique identifies parent and descended tasks and allocates them into the same VM. The file replication objective is to transfer a parent task's file replica to VMs where its descended tasks will be deployed. In this approach, the policy to apply one rule or the other is based on the file transfer time saved against the task execution.

Algorithm 4 presents the file reutilization and replication mechanism. The complete algorithm analyses all tasks in the workflow (lines 1-8). The bandwidth value is set to the minimum value among the machine holding the input files and a selected target VM (line 2). Line 3 presents the fundamental part of this algorithm. If the total time to transfer the input files exceeds execution time, then the task is added to the same task parent queue, e.g. reutilization. Otherwise the task is added to another VM, causing a new transfer (replication).

Algorithm 4 File reuse and replication

Input: Workflow *W*, *VM* set

Output: Reorganize $vm_i^{queue} \in \{VM\}$ with file reutilization

1: **For** all $t_i \in W$
2: $bandwidth = \min(vm_i^{bw} | t_i^{parents} \in vm_i^{queue}$,
3: **If** $\frac{in_i^{size}}{bandwidth} \geq \hat{t}_i^{total}$ **and** $t_i^{parents} = [\emptyset]$ **then**
4: $vm_i^{queue} \leftarrow t_i | t_i^{parents}$
5: **else**
6: $vm_i^{queue} \leftarrow t_i | t_i^{parents}$
7: **end**
8: **end**

Algorithm 5 Queue balance

Input: Workflow *W*, *VM* set

Output: Balanced $vm_i^{queue} \in VM$

1: $average\_queue = |W|/|VM|$
2: $bag = [\emptyset]$
3: **For** all $vm_i^{queue} \in VM$
4: $queue\_size = |vm_i^{queue}|$
5: **If** $vm_i^{queue} > average\_queue$ **then**
6: $bag \leftarrow [vm_{average_{queue}}^{queue}, vm_{queue\_size}^{queue}]$
7: **else if** $vm_i^{queue} < average\_queue$ **and**
8: $vm_i^{queue} \leftarrow [bag_1, bag_{average\_queue - queue\_size}]$
9: **End**
10: **end**

4.4.4 Queue balance

The objective of this technique is to balance all VM queues. Scheduler rules can overload a particular queue leading to unbalanced load across VMs. To balance scheduling queues, this procedure interchanges tasks between all queues in order to lower the difference between their loads without worsening any optimization goal. Algorithm 5 presents the balance technique. It first computes the average number of tasks between VMs (line 1). Line 2 empty the *bag* set. Later, lines 3–9 analyze all vm_i^{queue} for all VMs. Tasks are moved from overloaded queues, those with higher queue length than the average, to this *bag*. If a queue's size is lower than the average, tasks from *bag* are transmitted to it, considering its incurred data transfer time.

4.4.5 Workflow Contraction

BaRRS tries to group tasks of a workflow for faster distribution among VMs; it starts by grouping serial tasks. Figure 18 exemplifies this procedure; the serially connected tasks inside dotted ellipses are grouped together to produce a contracted workflow as shown on left side of the figure.

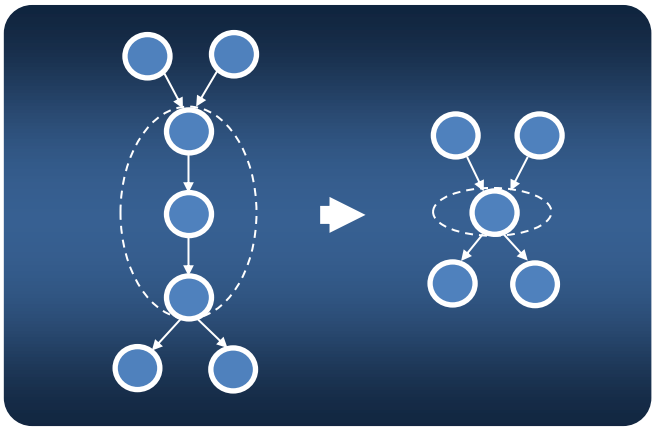


Figure 18. Workflow contraction example.

4.5 Experiment Setup

The performance of BaRRS was evaluated using a VMware-ESXi-based (version 5.5) private cloud to validate the solutions. This environment generates VMs matching AWS [15] instances such as the t2.small. The cloud consists of three Krypton Quattro R6010 with 4-way AMD Opteron™ 6300 series (64-Cores each), ESXi. Twelve VMs were prepared to perform as workers. Pegasus-WMS (4.2) on Ubuntu 14.04 was used as the workflow management system, where BaRRS was implemented.

Table 3. Characteristics of the scientific workflows.

	Nodes	*w-levels*	*parallel-tasks*	Average file size (MB)	Average task execution time (s)	Dependencies patterns
Epigenomics	100	8	24	749	2346	(2)(3)(4)
Montage	100	9	62	20.6	11.34	(2)(3)(4)
Cybershake	100	5	48	1156.1	51.70	(1)(3)(4)
Ligo	100	8	24	55.6	222.0	(1)(4)(5)
Sipht	100	7	51	22.02	210.27	(4)(5)

(1) Process; (2) Pipeline; (3) Data distribution; (4) Data aggregation; (5) Data Redistribution

Five scientific workflows were selected from [4] to produce the experiments. Workflows represent applications from different scientific areas including astronomy, geology, biology, cosmic analysis and biotechnology. Their details are presented in Table 3.

As described in Literature Review Chapter, most algorithms intended for cloud environment use a fixed number of VMs in their scheduling procedures. To the best of this researcher's knowledge, Provenance Adaptive Scheduling Heuristic [72], is among the state-of-the-art approaches that is also able to produce scheduling plans with different numbers and combinations of VMs based on execution runtime, monetary cost and reliability requirements. For this reason, the Provenance Scheduling approach is selected for comparison with BaRRS.

The Provenance scheduler analyses groups of tasks ready for execution, it groups them in queues with sizes depending on their historical execution time profile. This approach is able to increment the number of VMs as long as the monetary cost of an application does not exceed its upper limit (monetary constraint). The parameter values w_1, w_2 and α are set to 0.50. On the trade-off graphs, each point corresponds to a unique number of VMs expressed as "{}". For example, {3, 5, 7} represents trade-off points for VMs three, five and seven.

4.6 Results

Epigenomics workflow

The Epigenomics trade-off graphs in Figure 19 show how BaRRS outperformed Provenance. Provenance's low performance is related to the way it schedules tasks of a workflow: one task level at a time. This causes VMs to remain idle until an entire level finishes execution. Only then, VMs can continue to execute the next level task set. Furthermore, VMs do not save files if not used by the next executing task. As a result, files are sent to a central disk, thus adding unnecessary file transfers to increase both execution time and monetary cost.

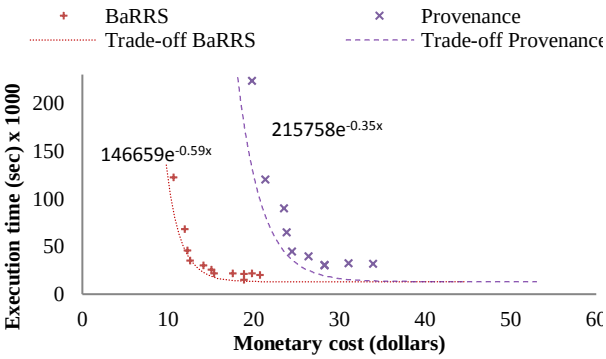


Figure 19. Epigenomics trade off.

Montage workflow

The main characteristic of the Montage trade-off graphs (Figure 20) is that most of the solutions are executed within one hour. This situation is caused by a low computing demand from Montage workflows. Tasks neither need high computing power nor large scale file transfer. It also presents a particular dependency pattern: each task on the second level depends on two tasks from the first level. BaRRS analyses both levels and explores data reuse by identifying the pair of parent tasks and their descendants. Following that, it groups and deploys them to the same VM. For this reason, BaRRS leads to lower execution and cost values in comparison to a Provenance approach.

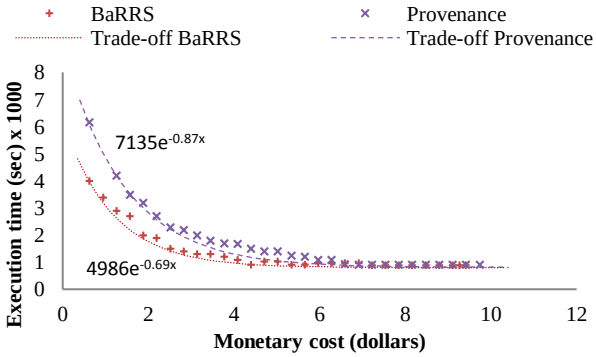


Figure 20. Montage trade-off frontier.

Cybershake workflow

An important characteristic of this workflow is that most of its solutions are executed at the average rate of 11,000 seconds, while not less than 10,000. The reason for such execution time is the need to transfer 80GB input files. The network transfer time of these files contributes to about 65% of total execution time or running time.

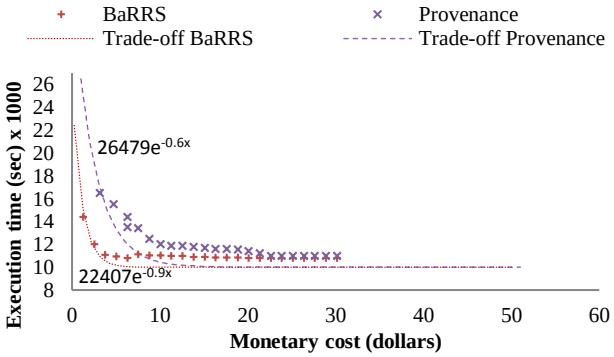


Figure 21. Cybershake trade-off graph.

Ligo workflow

Figure 22 presents the trade-off graph for BaRRS and Provenance. Solutions for both approaches tend to execute at an average execution time of 3000 seconds at a cost of 3.00 dollars. The reason for this behavior is the uniformity of the dependency patterns. The first two task levels contract (Workflow Contraction) in a single one, the same as levels four and five, converting the problem to a simple map of parallel tasks where four VMs is the correct number to achieve optimization constraints. Incrementing the number of VMs offers no execution time improvement while increasing the monetary cost.

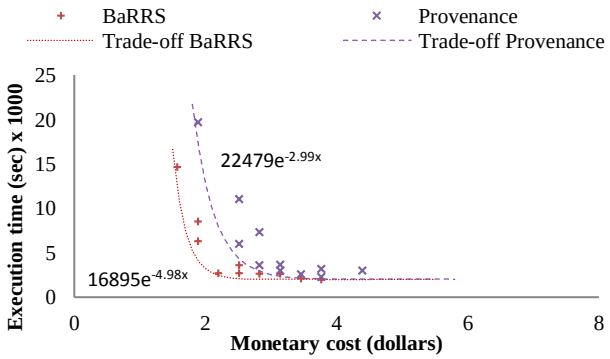


Figure 22. LIGO trade-off frontier.

4.7 Analysis - Optimization of Runtime and Monetary Cost

This section presents execution time, monetary cost and system utilization results for the complete range of |VM| configurations. This measurement is to analyze the total time VMs are active during execution, which is defined as:

$$Utilization = \frac{\sum_{j=1}^{|VM|}[vm_j^{time}]}{(Runtime)|VM|} \quad \text{Eq. 12}$$

Where,

$$vm_j^{time} = \sum_{i=1}^{\left|vm_j^{queue}\right|} \hat{t}_i^{exe} \quad \text{Eq. 13}$$

Epigenomics

The six-machine configuration presents the highest utilization value for BaRRS as shown in Figure 23. This condition is due to the following reasons. First, the Workflow's MaxLevel consumes 98.5% of the total execution time. Second, the size of files and makespan values are similar for all tasks. Third, the twenty-four tasks in this level can distribute uniformly on six VMs. For the same reason, this configuration presents a low execution time and monetary cost as shown in Figure 24 and Figure 25.

The execution times gradually decrease as the number of VMs increases (Figure 24). The reason for this behavior is the uniform distribution of tasks in the workflow graph. Furthermore, each task only depends on a single parent, allowing a very uniform task deployment across the VMs.

Montage

An important characteristic of the montage workflow is that the maximum number of tasks across the level is 62. For simplicity, this level is referred to as *MaxLevel*. Even though this level groups the majority of the tasks, it only contributes to about 57% of the total execution time. This factor causes solutions with higher utilization values to demand only a

small number of VMs. Furthermore, as the number of VMs increases, their utilization significantly drops (Figure 27).

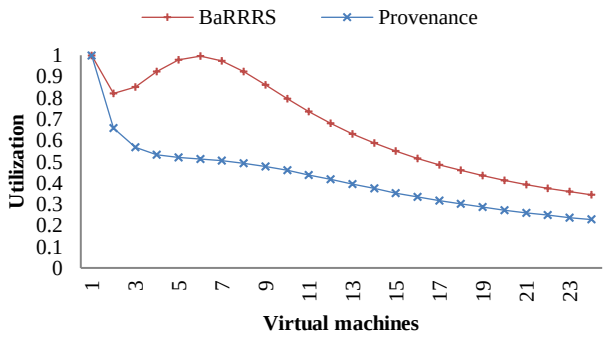


Figure 23. Epigenomics system utilization.

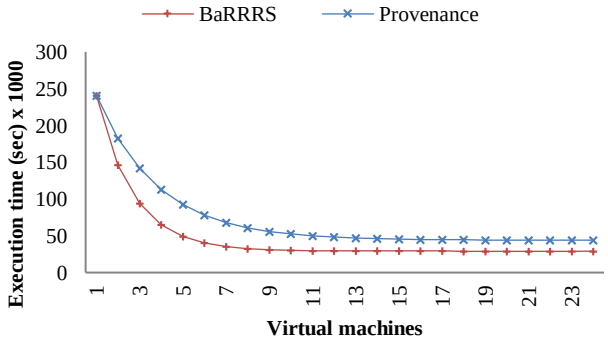


Figure 24. Epigenomics execution time.

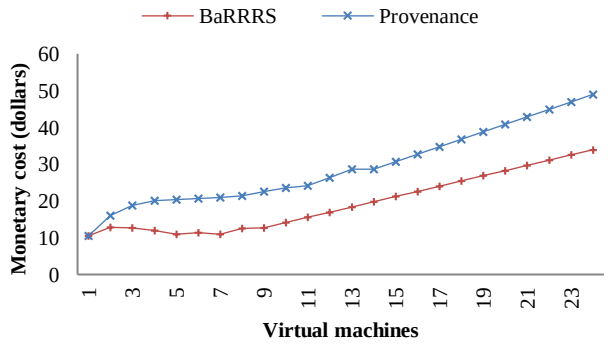


Figure 25. Epigenomics monetary cost.

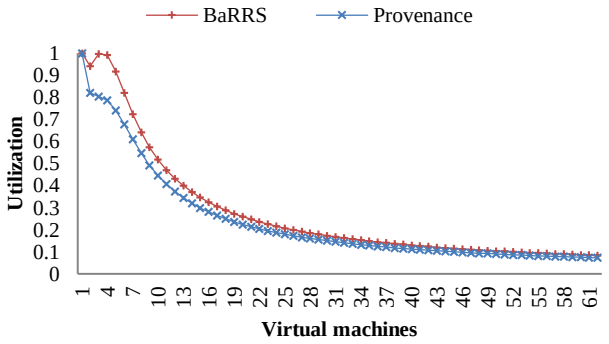


Figure 26. Montage system utilization.

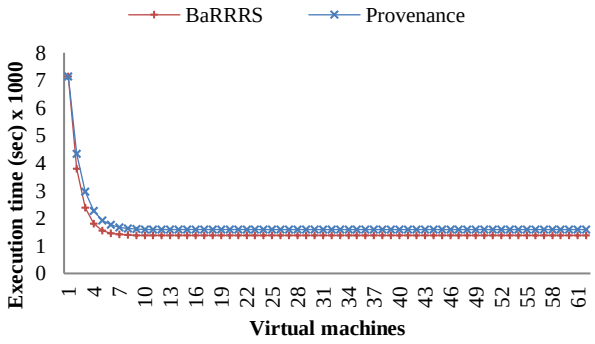


Figure 27. Montage execution time.

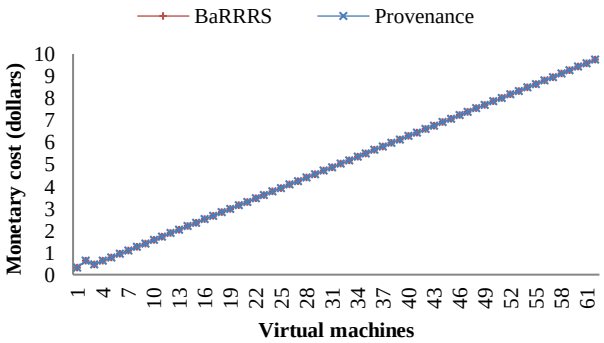


Figure 28. Montage monetary cost.

Cybershake

Cybershake solutions have considerable low system utilization as shown in Figure 29. The main reason for this performance is the size of input files. The average transfer input time is about 92.4 seconds, while task average execution time is about 51.7 seconds. This data-intensive workflow is suitable to execute on a small number of VMs to obtain higher utilization values. Moreover, users are able to evaluate whether they execute their workflow on the cloud or on their own resources based on this analysis. This highlights the importance of this study to guide users.

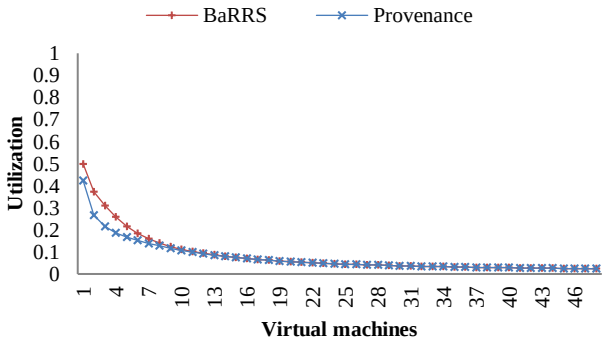


Figure 29. Cybershake system utilization.

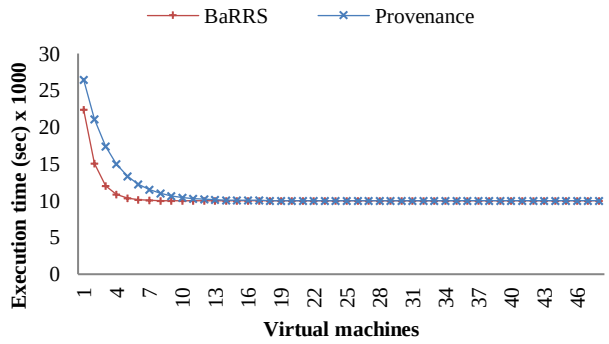


Figure 30. Cybershake execution time.

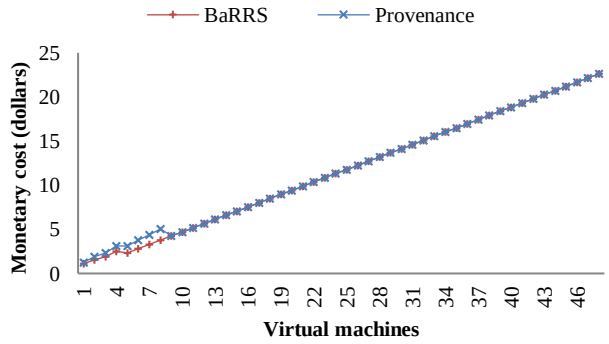


Figure 31. Cybershake monetary cost.

LIGO

LIGO has two MaxLevels with 24 tasks. All tasks in these two groups demand different file sizes with different execution times. This causes lower system utilization values for Provenance experiments (Figure 32). Both BaRRS and Provenance approaches lead to similar results for execution time and monetary cost as presented in Figure 33 and Figure 34, mainly because of fairly equal file transfer and execution times for tasks. Nevertheless, BaRRS still outperforms, though marginally, Provenance because it considers all tasks during scheduling. However, BaRRS contributes with the highest utilization value of 97% while

Provenance's highest value is 75% as shown in Figure 32. This improvement is also reflected in the execution time and monetary cost for solutions with 4 VMs as shown in Figure 33Figure 34. The data replication technique of BaRRS caused those encouraging results. In contrast, Provenance groups tasks based on their computational demands without a deep analysis of file transfers causing unsatisfactory results.

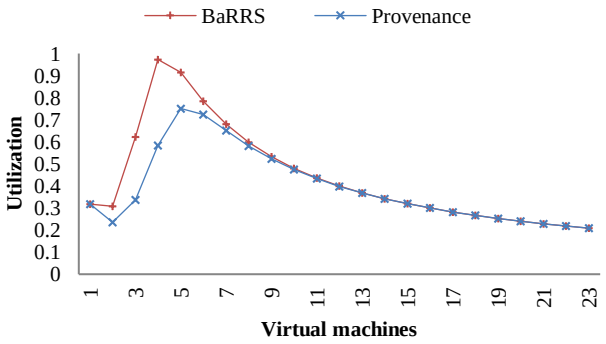


Figure 32. LIGO system utilization.

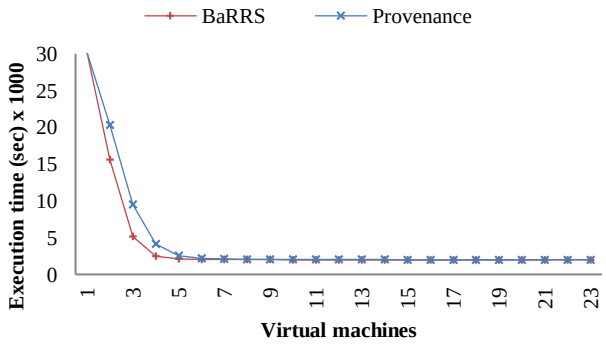


Figure 33. LIGO execution time.

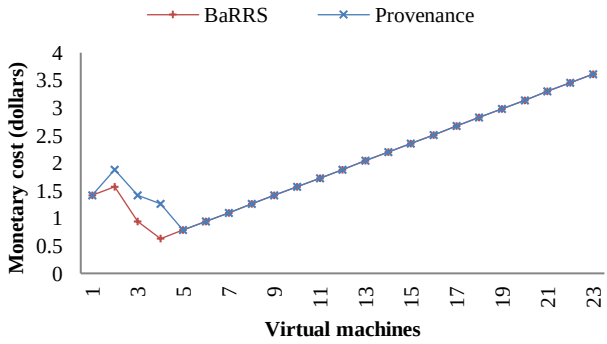


Figure 34. LIGO monetary cost.

4.8 Summary

This chapter presented the BaRRS scheduling approach for deploying scientific workflows on cloud-based VM resources. BaRRS is based on three techniques and a deep workflow analysis. It produces a scheduling configuration that gives an application owner the flexibility to choose different combinations of VMs based on execution time and monetary cost tradeoff. These techniques include queue balancing, file reuse, and file reutilization. The BaRRS approach takes special consideration of the workflow feature analysis such as file sizes, task parallelism, and task interdependencies. Four scientific workflows are selected as the application benchmark to test BaRRS performance. BaRRS was compared against the state-of-the-art Provenance scheduling approach, experiments proved BaRRS's superior performance in meeting conflicting requirements.

5 GA-ETI: A Genetic Algorithm to Select the Number of Resources to Execute Workflows

5.1 Preliminaries

This chapter presents GA-ETI (Genetic Algorithm Scheduler with Efficient Tuning of resources), an algorithm for selecting a number of VMs that achieves a minimal execution time and monetary cost compared with present scheduling techniques. GA-ETI employs the distribution mechanisms of BaRRS and extracts the capabilities of the genetic algorithm to produce optimal scheduling solutions.

Firstly, Section 5.2 presents the environment model to run GA-ETI. Then Section 5.3 describes the problem statement. Subsequently, Section 5.4 describes in full detail the GA-ETI approach. Next, Section 5.5 presents the experimental setup to test the GA-ETI. Section 5.6 provides a discussion of the obtained results focusing on GA-ETI performance. Section 5.7 gives an analysis of the selection of number of resources. Then Section 5.8 gives a discussion on the algorithm performance. Finally, Section 5.9 provides a summary with the most remarkable features of experimentation and algorithm performance.

The GA-ETI main contributions are: (1) a framework to solve the problem to select the number of resources to run applications in cloud computing systems, (2) an adapted GA

scheduler capable of providing superior results in terms of execution time and monetary cost when compared with up-to-date schedulers, (3) an adaptation of the GA without the excessive randomness presented in the original genetic algorithm. The core of this chapter has been sent for publication to the *Journal of Computational Science* [2].

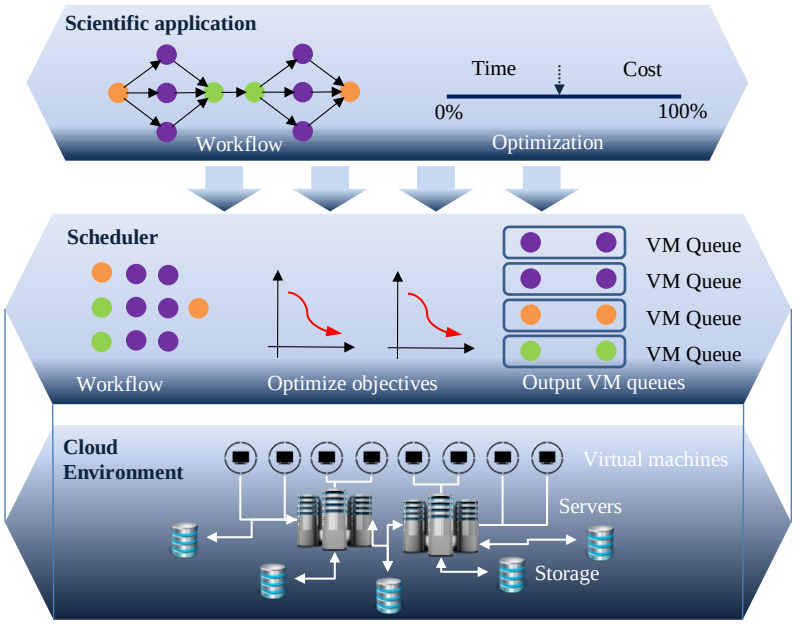


Figure 35. Architecture for scientific workflow execution.

5.2 Framework

This section presents model and parameter definitions in order to define the scheduling problem in the next section. In this study, the architecture for scientific workflow execution in cloud environments is divided into three layers: (1) a **Scientific Application** aims to be run on a cloud system, optimizing runtime and monetary cost; (2) a **Scheduler** engine acting as a connection between the cloud environment and the scientific application, its goal is to distribute a workflow's tasks; and (3) a **Cloud Environment** containing a group of servers offering VMs on a pay-as-you-go basis. An illustration of the described framework is presented in Figure 35. The first layer, Scientific Application, considers a workflow $\boldsymbol{W}$ aiming to be executed in a cloud system where a user is able to indicate optimization levels for execution time and monetary cost, w_1 and w_2 respectively. The second layer, Scheduling, receives workflow description, analyzes it and distributes tasks among the available resources assembling a queue for each VM. Finally, a Cloud Environment is a physical place hosting a

group of servers and storage devices providing computing power to clients through virtualization. Each client has the option to select a number of resources to build his/her particular group of resources $\boldsymbol{VM}$.

This model contemplates six assumptions: (1) scheduler analyzes and executes one workflow at a time, (2) scheduler accepts both computing and data-intensive workflows, (3) every VM has a fixed bandwidth (vm_j^{bw}), number of cores (vm_j^{cores}), cost per quantum of time (vm_j^{cost}), memory size (vm_j^{mem}) and disk size (vm_j^{disk}), (4) our proposed scheduler, GA-ETI, negotiates with the cloud provider to obtain the required VMs prior to execution of each workflow, (5) users must supply or estimate execution time $\hat{t}_i^{exe}$ for every task of W, as envisioned on Pegasus-WMS, and (6) resource deployment assumes each VM executes one task at a time.

This chapter uses part of the model from the work presented in [1], a research focusing on the balancing of task queues in the execution of scientific applications in cloud environments. The specific modifications to its model contributed in enhancing the output of this research: firstly, the term $idle_{vm}^{task}$ was introduced to decrease the overhead scheduling time generated when calculating the execution of tasks from a given VM. This term comprises the execution time from all parent tasks from a particular task. It is embedded in the vm_j^{queue} for time calculation purposes only. In this sense, the model avoids recurrent execution time calculation whenever it finds task interdependency with other VMs; as a result analysis of a complete scheduling configuration time obtained a reduction of 40% in the study's experimental practice. Secondly, a resource utilization constraint is included in order to procure an efficient usage of resources. For its experiments, this study takes only the scheduling configurations with the highest utilization resource values. Figure 36 presents an example for the calculation of utilization for a set of three VMs. Resource vm_3 remains busy for two hours while vm_1 and vm_2 execute their loads in 1.8 hr. For this reason, the cloud provider charges the user for two hours for each machine causing a utilization resource of 0.933.

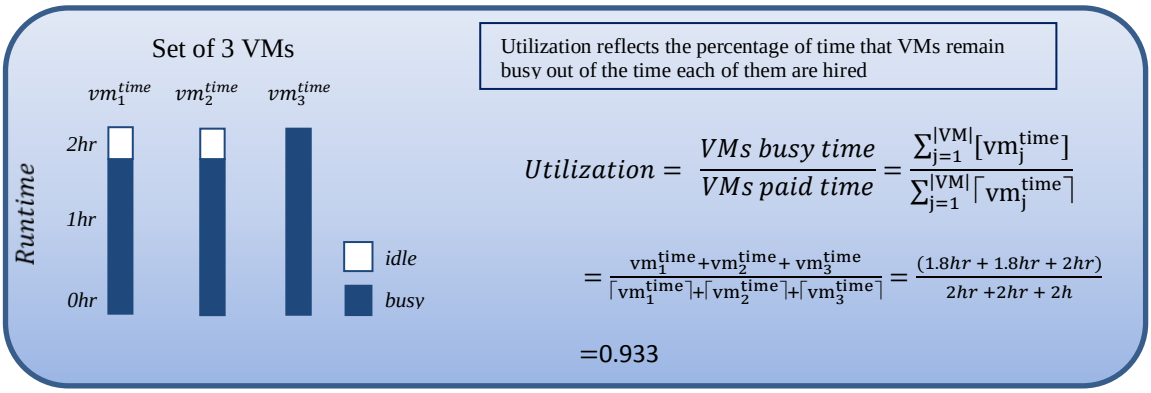


Figure 36. Resource utilization example.

5.3 Problem Statement

Scientific Workflow Scheduling (SWS) is a problem defined as assigning tasks to virtual machines to minimize: (1) total makespan to execute all workflow tasks and (2) monetary cost that a user pays to have his/her application completed. To formally express this problem, assume tasks are clustered and assigned to several VM queues $\{vm_1^{queue}, vm_2^{queue} \dots vm_{|VM|}^{queue}\}$ to be executed by $\{vm_1, vm_2 \dots vm_{|VM|},\}$ respectively. A cluster of tasks is defined as a decomposition of a workflow's tasks set into disjoint subsets of which the union is the original set. For instance, a pool of |VM| = 2 machines with $vm_1^{queue} = \{t_1, t_2\}$ and $vm_2^{queue} = \{t_3, t_4\}$ is executing a workflow of four tasks $W = \{t_1, t_2, t_3, t_4\}$.

Given the above description, an SWS problem is stated as defining a pool of resources $\boldsymbol{VM}$ and assigning a workflow's ($\boldsymbol{W}$) tasks to each virtual machine to minimize their execution time and monetary cost:

$$MSpn = LFT_{j=1}^{|VM|}[vm_j^{time}] \quad \text{Eq. 14}$$

$$MCst = \sum_{j=1}^{|VM|} \lceil vm_j^{time} \rceil vm_j^{cost} \quad \text{Eq. 15}$$

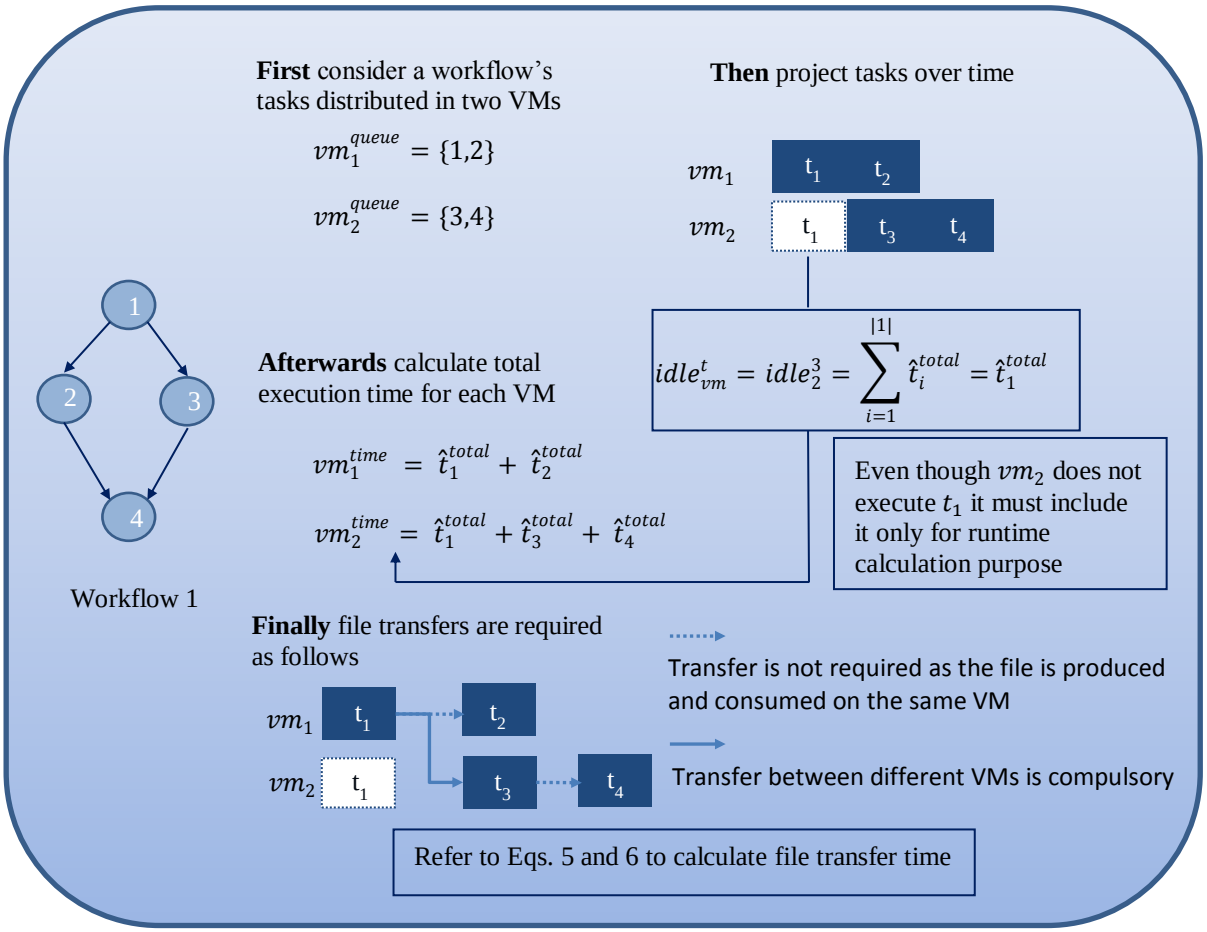


Figure 37. Example of runtime calculation for a 4-task workflow.

Makespan, Eq. 14, also referred to as runtime and execution time through the text, is the value of the LFT (Latest Finishing Time) from all the VMs executing workflow W, while monetary cost is the sum of all VMs cost multiplied by their respective round up runtime to the closest integer as expressed in Eq. 15. In order to reduce monetary cost presented in [1], this problem statement allows machines to be launched at different times by the cloud provider. With this modification the proposed SWS solver has the freedom to employ a particular VM for a specific interval. This modification led to a resource utilization improvement of up to 30%. Similarly, $MCst$ only considers the amount of time each machine is hired, yet cloud providers charge an hourly rate. Eq. 16 expresses the time each VM takes to execute its corresponding load, where vm_j^{queue} refers to the list of tasks assigned to vm_j. In order to reduce the scheduling overhead time presented in [1], this study introduces the term $idle_{vm}^{task}$ (Eq. 17, exectuion time of parent task) in the calculation of vm_j^{time}. In this sense, every vm_j^{queue} would contain the complete information from all tasks to calculate its total runtime and monetary cost without waiting for its calculation on a different vm_j^{queue}.

This will enable the SWS solver to reduce the overhead time at the scheduling stage. An illustration of vm_j^{time} calculation is presented in Figure 37.

$$vm_j^{time} = \sum_{i=1}^{|vm_j^{queue}|} \left[\hat{t}_i^{total} + idle_{vm}^{task} \right] \quad \text{Eq. 16}$$

$$idle_{vm}^{task} = \sum_{i=1}^{|parents|} \hat{t}_i^{total} \quad \text{Eq. 17}$$

The value of $\hat{t}_i^{total}$ expressed in Eq. 18 contemplates the time to transfer the required n number of files and the time to run a task's executable program, $\hat{t}_i^{exe}$, which is a value provided by the user

$$\hat{t}_i^{total} = \sum_{f=1}^{n} \hat{t}_i^{f} + \hat{t}_i^{exe} \quad \text{Eq. 18}$$

Finally, the time to transfer each file, $\hat{t}_i^{f}$, depends on the bandwidth of the VM's parents vm_p^{bw} and vm_i^{bw}, expressed as:

$$\hat{t}_i^{f} = \frac{f_i^{size}}{min\left(vm_p^{bw}, vm_i^{bw}\right)} \quad \text{Eq. 19}$$

5.4 GA-ETI Approach

The Genetic Algorithm (GA) is a metaheuristic motivated by genetic evolution with important features for combinational optimization. It is a robust technique to solve complex problems in engineering and science due to its ability to detect a global optimum in the complete search space [41]. Contrary to current heuristics solutions [6, 73, 74] GA does not build a single solution. Instead, it applies genetic operators to current configurations (parents)

with the objective of generating stronger solutions (offspring) from evolution [42]. For these reasons, this study modified the GA into the GA-ETI to solve the SWS problem. This section presents the fundamentals of the GA and its adaptation for the cloud scheduling problem.

5.4.1 Genetic Algorithm (GA)

The original GA has an initial population that starts the GA with a group of possible solutions [45]. Each chromosome is a string of genes encoding a specific solution. The particular nature of an optimization problem defines chromosome and gene characteristics. Through the genetic process, GA selects fittest chromosomes, combining them to produce a final strong solution. The first phase to produce a new population is the selection operator. Its objective is to select chromosomes to produce the next population [42]. A frequently used selection technique is the roulette wheel where each chromosome is allocated a portion of the wheel according to its fitness value; hence chromosomes with greater values are allocated more slots with more chances to be selected for the next population. Then, the genetic operators combine chromosomes to hopefully produce chromosomes with higher fitness values: (1) Crossover splits and combines genes between two selected chromosomes according to a predefined probability; (2) Mutation randomly selects genes from a chromosome and changes their values according to another predefined probability. Additionally, the fittest chromosomes are directly copied to the next population. Finally, GA terminates when it meets selected criteria. The most used criteria are total execution time, the number of iterations, fitness value, and conditional minimum improvement [41, 42, 45, 47]. The fitness function in GA evaluates the quality of each chromosome. For maximization problems, the fitness function is proportional to the problem cost function while minimization problems use the inverse value of this equation.

5.4.2 The GA-ETI

This section presents the enhanced Genetic Algorithm with Efficient Tune-In of resources inspired by the fundamentals of the genetic process.

GA-ETI's objective is to (1) solve the scheduling problem and (2) return the number of resources a particular workflow needs for execution. Concurrently, GA-ETI optimizes monetary cost and execution time. Genetic operators are carefully adjusted to distribute a workflow's tasks on VM queues. Algorithm 1 presents the GA-ETI with its featured

components. Firstly, in step 1, GA-ETI uses Algorithm 2 to generate a preliminary population with a size *IP* greater than a regular population. Then, step 2 reduces this preliminary population to a regular size selecting the fittest chromosomes. Steps 4 to 14 develop the main loop. Step 5 evaluates every chromosome using Eq. 20. Then, in step 6 a quarter of the fittest chromosomes are directly copied to the next population for elitism and the rest of the chromosomes are selected using the roulette wheel. Afterwards, steps 7-9 apply one-point and multiple-point crossover to a selected group of chromosomes to produce offspring followed by a mutation operator in steps 10-13. The algorithm stops when neither $MSpn$n or $MCst$ present an improvement by returning chromosomes with the highest fitness value (step 15).

Algorithm 1: GA-ETI

Input: Workflow *W*, *VM* set
Output: Scheduling plan

1: **Generate** preliminary population (Algorithm 2)
2: Initial population ← Select fittest chromosomes
4: **While** time **or** cost **still improve**
5: Evaluation
6: Selection
7: **Crossover**
8: - Conventional crossover
9: - Clustered crossover
10: **Mutation**
11: - Swap
12: - Increment VMs
13: - Decrement VMs
14: **End**
15: $Fittest\ Solution = Max_{i=1}^{total} F_{fitness}$

$$F_{fitness} = w_1 \frac{(max^{MSpn} - MSpn)}{(max^{MSpn} - min^{MSpn})} + w_2 \frac{(max^{Cst} - MCst)}{(max^{Cst} - min^{Cst})} \quad \text{Eq. 20}$$

The objective of $F_{fitness}$ (Equation 20) is to combine makespan and economical cost into a single equation. For accomplishing this requirement, each member in $F_{fitness}$ is forced to produce values between 0 and 1 as expressed in:

$$0 < \frac{(max^{MSpn} - MSpn)}{(max^{MSpn} - min^{MSpn})} < 1 \text{ and } 0 < \frac{(max^{Cst} - MCst)}{(max^{Cst} - min^{Cst})} < 1$$

Where a value of 1 represents a solution with the best (lowest) registered makespan or monetary cost and a value of 0 represent a solution with the worst (highest) ever registered value. By forcing each member in producing values between 0 and 1, $F_{fitness}$ is able to combine both quantities i.e. makespan and monetary cost. Then the terms w_1 and w_2 give an optimization priority from 0 to 1 for each optimization objective by limiting their values as follows:

$$w_1 + w_2 = 1$$

Chromosome and fitness function description

Chromosomes represent a complete workflow scheduling where each gene represents a task and the required VM to execute it; hence, chromosome length equals the size of the given workflow |W|. Figure 38 describes a chromosome with an example. On it, the position of $gene_1$ represents $task_1$, $gene_2$ represents $task_2$ and so on. Similarly, the value of $gene_1$, 1, expresses that vm_1 executes the represented task, in this case $task_1$; value of $gene_2$, 1, assigns $task_2$ to vm_1 and so on.

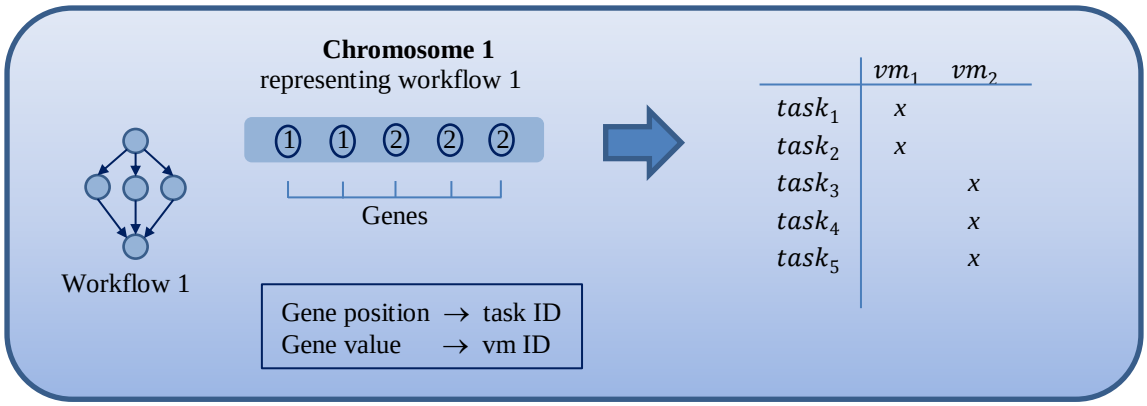


Figure 38. Chromosome representation in GA-ETI.

Eq. 20 assigns a fitness value to each chromosome based on its makespan and monetary cost on every iteration of Algorithm 1 (steps 4 - 14). $F_{fitness}$ keeps a record of maximum (max^{MSpn} and max^{Cst}) and minimum ($min^{MSpn}$and min^{Cst}) values of $MSpn$ and $MCst$ in order to provide a global evaluation to each solution; these values update on each iteration on the main loop of GA-ETI. Additionally, the fitness equation enables the user to

assign priority to a given optimization objective employing w_1 and w_2 as time and cost optimization weights respectively where $w_1 + w_2 = 1$.

Pre-initial population

Algorithm 2 leads to an initial population with fittest chromosomes for GA-ETI. This algorithm first produces a larger initial pre-population, then it reduces the population selecting the best chromosomes to build the first generation. Algorithm 2 firstly identifies the number of genes, largest-level and parallel-tasks and then assigns a random VM to each task in steps 1-3. Since workflows do not require an unlimited number of resources, the algorithm limits the size of $\boldsymbol{VM}$ to parallel-tasks which are the maximum number of tasks that can run in parallel. The main loop in steps 4-13 executes IP times to build the pre-initial population. The loop in steps 7-10 assigns a random value to each gene on every chromosome with values from 1 to parallel-tasks. Finally, step 12 returns the initial population of a regular size. Founded on this study's practical tests, the best results were obtained with an IP value of 10.

Algorithm 2: Creating the Initial Population

Input: Workflow *W*
Output: Initial population

1: *genes* = number of tasks on workflow
2: *largest-level* = Largest workflow level
3: *parallel-tasks* = number of the tasks in *largest-level*
4: **Set** *IP*
5: **For** j=1:IP
6: **For** k=1: size of population
7: **For** i=1: genes
8: $gene_i$ = random value from [1 to *parallel-*
9: $chromosome_k \leftarrow gene_i$
10: **End**
11: **End**
12: pre_initial_population $\leftarrow chromosome_k$
13: **End**
14: **Return** pre_initial_population

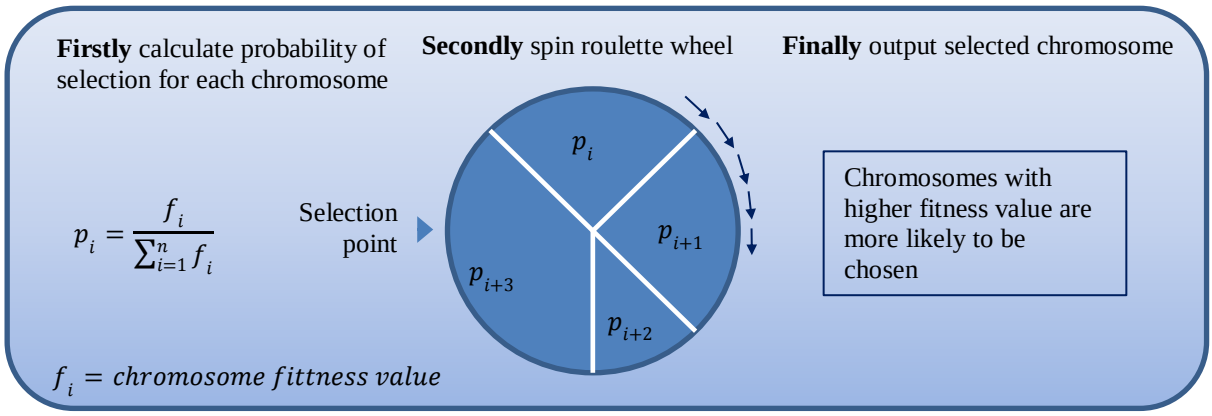


Figure 39. Roulette wheel illustration.

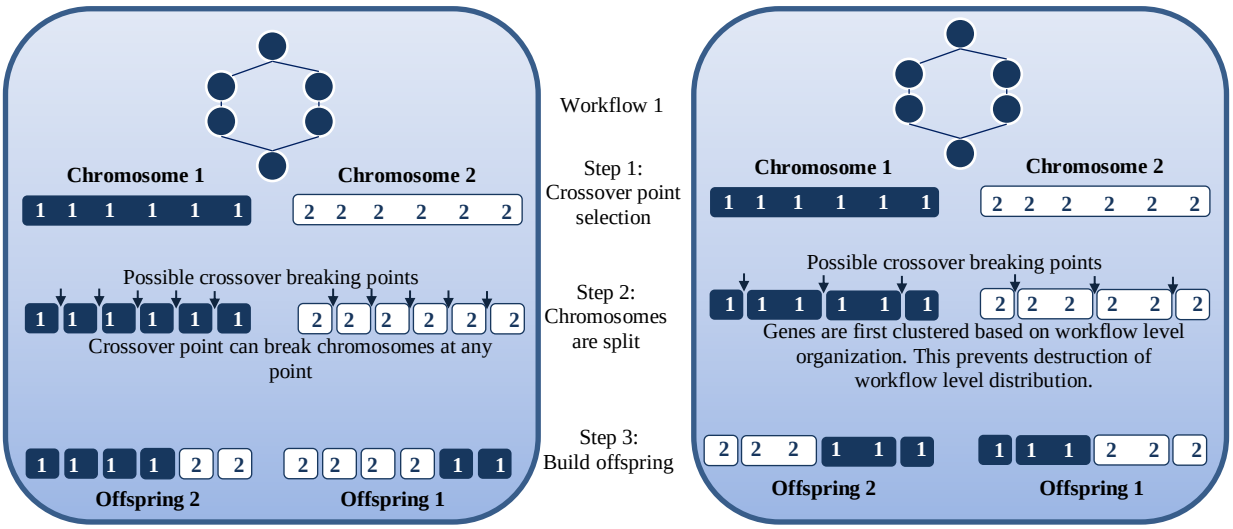
5.4.3 Genetic Operators: Selection, Crossover, Mutation

GA-ETI makes use of the roulette wheel for selection of chromosomes for genetic operators. The roulette wheel is a selection process simulating a partitioned spinning wheel. Partition size depends on fitness of its elements. In this study, each element is a chromosome and its partition size depends on its fitness value. Figure 39 presents a graphical description of the roulette wheel. First it assigns size (p_i) of each partition to every element (chromosome). Then, all partition sizes are assigned to the wheel. Finally, the roulette wheel spins and selects a winner.

GA-ETI adapts the conventional crossover and swap mutation from the original GA to be used in this model. Additionally, a modified crossover and new increment and decrement mutation operators were designed and added to the GA-ETI to produce a powerful tool. Description of these mechanisms is as follows.

Conventional crossover

This operator is the accurate adaptation of the original GA crossover into the scheduling problem. It allows the breaking of a pair of chromosomes into a limited number of pieces and then combining their parts in order to produce offspring. Number and location of breaking points are chosen arbitrarily. Figure 40.a presents an instance of this process. Firstly, chromosomes 1 and 2 are selected from the population using the roulette wheel. Then, step 2 highlights that chromosomes can break on any number and location; in this case, only one crossover point is used, dividing chromosomes into two parts each. Finally, chromosomes are combined building offspring 1 and 2.



a) Conventional crossover operation

b) Clustered crossover operation

Figure 40. Crossover operation example.

Clustered crossover

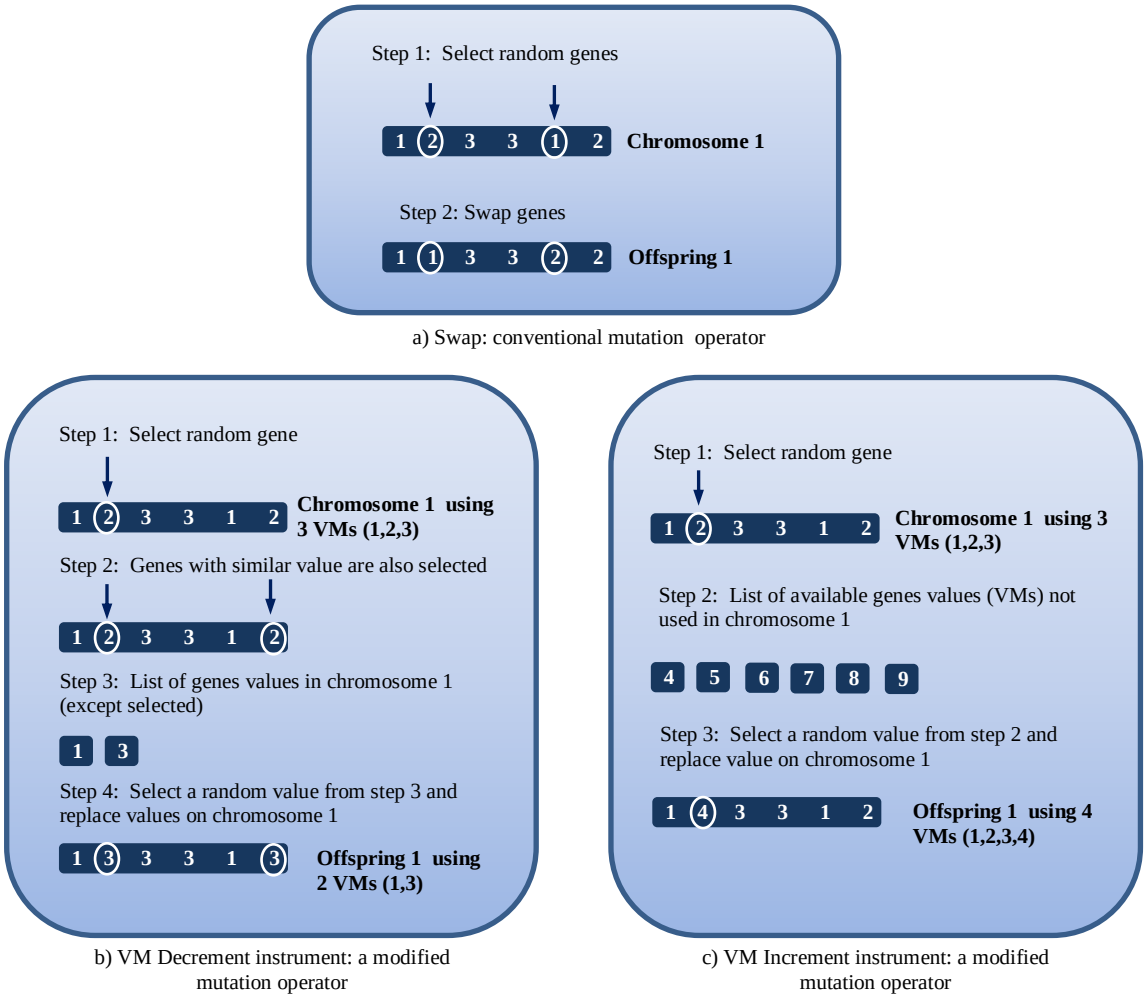
This enhanced crossover operator is specially adapted to the scheduling problem. While conventional crossover breaks chromosomes at any location, clustered crossover does not separate genes from the same workflow. This procedure allows GA-ETI to produce newborns combining clusters of genes representing workflow levels. Figure 40.b presents an example of this procedure. It first selects a pair of chromosomes in step 1; then, step 2 presents clusters of genes for each chromosome. Workflow 1 is split into four-cluster chromosomes, each one representing a level from the given workflow. From this chromosome division is then selected a crossover breaking point(s). Finally, chromosomes mix with each other, producing offspring.

Swap mutation operator

This study adapts the original GA swap operator to be applied with the GA-ETI. The swap operator produces an offspring from a single chromosome, it first selects a pair of genes and then it swaps their values. A pair of gene values is interchanged in each swap operation. Figure 41.a presents an example of this operation. In step 1, a random pair of genes is selected from the parent chromosome. Then in step 2, the selected genes swap their values, producing offspring 1.

Increment and decrement mutation operators

Increment and decrement instruments are a modification of the mutation operator to change the number of VMs that a given chromosome uses. Figure 41b-c explain these operators with an example. The decrement process in Figure 41b reduces the number of VMs, i.e. gene values. In this example, chromosome 1 has three different gene values (1, 2 and 3) while offspring 1 ends up with only two different values of genes (1 and 3). The procedure starts with step 1, it first selects a random gene, and then it selects every gene with a similar value. In step 3, it lists the different gene values presented on chromosome 1. Finally, in step 4, it selects a random value from the list in step 3 and replaces the selected gene(s) from step 2. As for the increment operator in Figure 41c, it adds a new gene value, i.e. a new VM to the chromosome. As this example shows, offspring 1 ends up with an additional gene value. This operator first selects a random gene value in step 1. Then in step 2, it lists the available VMs that GA-ETI can use, but are not part of chromosome 1, in this particular case 4 – 9. Finally, in step 3, a random value from the mentioned list replaces the selected gene from step 1.



a) Swap: conventional mutation operator

b) VM Decrement instrument: a modified mutation operator

c) VM Increment instrument: a modified mutation operator

Figure 41. Mutation operators for GA-ETI.

5.4.4 GA-ETI Algorithm Complexity

In order to measure GA-ETI's complexity, this study defines the growing order of the algorithm. For this purpose, let us first define $T(t,s,l)$ as the number of time units GA-ETI needs to produce a scheduling configuration of a given workflow. GA-ETI is divided into five different stages: initial population generation, evaluation, selection, crossover and mutation. The complexity of each stage is extracted from Table 4, hence $T(t,s,l)$ obtains the value of $(t)(s)(IP) + (s) + (s-l) + (0.6)(s)c + (0.3)(s)$. Since values of IP, l and c are constants and s does not depend on the number of tasks then $T(t,s,l) = O(t)$ where $O(t)$ expresses the growing order of GA-ETI as a linear function of the number of tasks in the workflow.

Table 4. Parameter description to determine GA-ETI algorithm complexity.

Stage	Complexity
Generate initial population	
Assign a random number [1:*VM*] to each gene (t) from the complete population (s) augmented	$(t)(s)(IP)$
Evaluation	
Calculates $F_{fitness}$ for each chromosome	(s)
Selection	
Run roulette wheel (p) times to build a new population taking off number of chromosomes from elite operator	$(s-l)$
Crossover	
The complete population has a maximum probability of 0.6 to go through crossover operator where each operation depends on a constant number of crossing points	$(0.6)(s)(c)$
Mutation	
The complete population has a maximum probability of 0.3 to go through mutation operator where the number of interchangeable genes remains constant	$(0.3)(s)(c)$

(t) Number of tasks (genes); (s) Size of population (number of chromosomes); (IP) Pre-initial population factor; (l) Elite size group; VM Maximum number of virtual machines; (c) Constant

As readers will notice, the number of VMs does not affect complexity since it is only considered as a pool of values where genes initially obtain their identification number (see stage 1 in Table 4). Furthermore, the size of population s appears at every stage but is not affected by type and size of workflow. Experimentation also reveals that the number of iterations is affected by neither workflow type or size even for the unmodified GA. The growing order of GA-ETI depends only on the number of tasks.

5.5 Experiment Setup

To evaluate the performance of GA-ETI, a private VMware-vSphere (version 5.5) private cloud was employed to validate the solutions. Virtualization in this environment generates instances matching the AWS [15] t2.small VM. The cloud consists of three Krypton Quattro R6010s with 4-way AMD Opteron 6300 series (64-Cores each). For system management, Pegasus-WMS (4.2) was employed on Ubuntu 14.04 where GA-ETI was implemented with the parameters shown in Table 5. The five scientific workflows presented in Section 4.2 are used to gauge the efficiency of the specific scheduling approach in this work. The inputs for the experiments are workflow files including (i) executable files, (ii)

data files and (iii) workflow dependency description. The goal of experiments is to test the proposed scheduler and analyze its behavior against up-to-date scheduling algorithms.

This study compares GA-ETI against three up-to-date schedulers in the same field. This algorithm selection includes: Provenance [72], HEFT [13] and FSV (Flexible Selection of VMs) [7]. In summary, Provenance groups tasks on queues depending on their historical execution time and file sizes; HEFT creates a pre-schedule queue based on a critical path and then distributes tasks following an earliest finishing time; and FSV emulates HTCondor's behavior [75] to execute tasks on available VMs. To manage the number of VMs, Provenance increments the number of VMs as long as the monetary cost does not exceed a user's budget; HEFT and FSV use as many VMs as are available.

Table 5. GA-ETI setup for population and genetic operators

Parameter	Symbol	Value
Crossover probability	p_c	0.60
Swap mutation	p_s	0.25
Increment mutation	p_{inc}	0.20
Decrement mutation	p_{dec}	0.20
Pre-initial population factor	IP	10
Initial population	P	500
Time weight constraint	w_1	0.5
Cost weight constraint	w_2	0.5

In this section, the four algorithms schedule the workflows described in Section 4.2 . For this first experimental stage, scheduling algorithms have access to as many VMs as parallel tasks in the workflow. For instance, LIGO is able to use a pool of 24 VMs (see parallel-tasks on Table 3).

5.6 Results

Table 6 presents the obtained results. For the Epigenomics workflow, GA-ETI and HEFT produce similar runtime results (21190 and 22890 seconds, respectively) as its nodes have a very uniform distribution allowing schedulers to allocate tasks evenly among VMs. In contrast, FSV and Provenance presented higher time values (67325 and 89011 seconds, respectively), on one hand FSV allocates tasks to any available VM without considering dependencies causing duplication of data files, on the other hand, Provenance has an internal grouping offset value that groups tasks based on previous executions and not on current tasks. As for the Cybershake workflow, it presents a simple dependency pattern among tasks allowing FSV to obtain similar results as GA-ETI and HEFT; in contrast, Provenance is

prevented from delivering better results due to its task grouping policy. Montage workflow highlights the need to analyze dependencies between tasks; for this workflow, GA-ETI's scheduling policy allows groups of tasks sharing a common parent task to be allocated to the same VM in order to lower file transfer time. In summary, HEFT outperformed Provenance and FSV due to its simplistic nature in allocating tasks into VMs, even though HEFT does not analyze job dependencies which prevents it from delivering lower values for time and monetary cost as exhibited by GA-ETI.

Table 6. Execution time and monetary cost results for FSV, GA-ETI, HEFT and Provenance.

	Epigenomics	**Cybershake**	**Sipht**	**Montage**	**Ligo**
	Time (s)	Time (s)	Time (s)	Time (s)	Time (s)
GA-ETI	**21190**	**4619**	**3587**	**270**	**3486**
HEFT	22890	5199	3687	385	4717
FSV	67325	6549	6106	475	8508
Provenance	89011	8711	5090	550	8340

5.7 Analysis – Selection of Number of Resources

This section selects the best configuration from FSV, GA-ETI, HEFT and Provenance from the results exhibited in Figure 43. To that end, each algorithm is forced to produce a scheduling configuration for every possible number of VMs, then the outcomes are evaluated with $F_{fitness}$ for each algorithm. Once all configurations are rated, the highest value is selected for each algorithm. Final results including number of VMs are presented in Table 7 and Figure 42 shows execution time and monetary cost.

Table 7. Optimal number of VMs for HEFT, Provenance, FSV and GA-ETI.

	HEFT	**Provenance**	**FSV**	**GA-ETI**
Sipht	6	6	5	5
Cybershake	3	7	4	3
Epigenomics	6	8	5	24
Ligo	8	18	9	5
Montage	5	6	5	4

Results show that the number of tasks in a workflow does not influence the final number of resources a given application needs. The number of VMs is related to (1) workflow computational requirements, (2) file transfer demands, and (3) task dependency constraints. For each workflow, all approaches select a similar number of resources with only two specific exceptions for GA-ETI on Epigenomics and Provenance for Ligo. For the first

case, GA-ETI converges to solutions employing as many VMs as the number of tasks on the largest workflow level (24). For the second case, Provenance selects a high number of VMs due to the high number of tasks it groups on its first level.

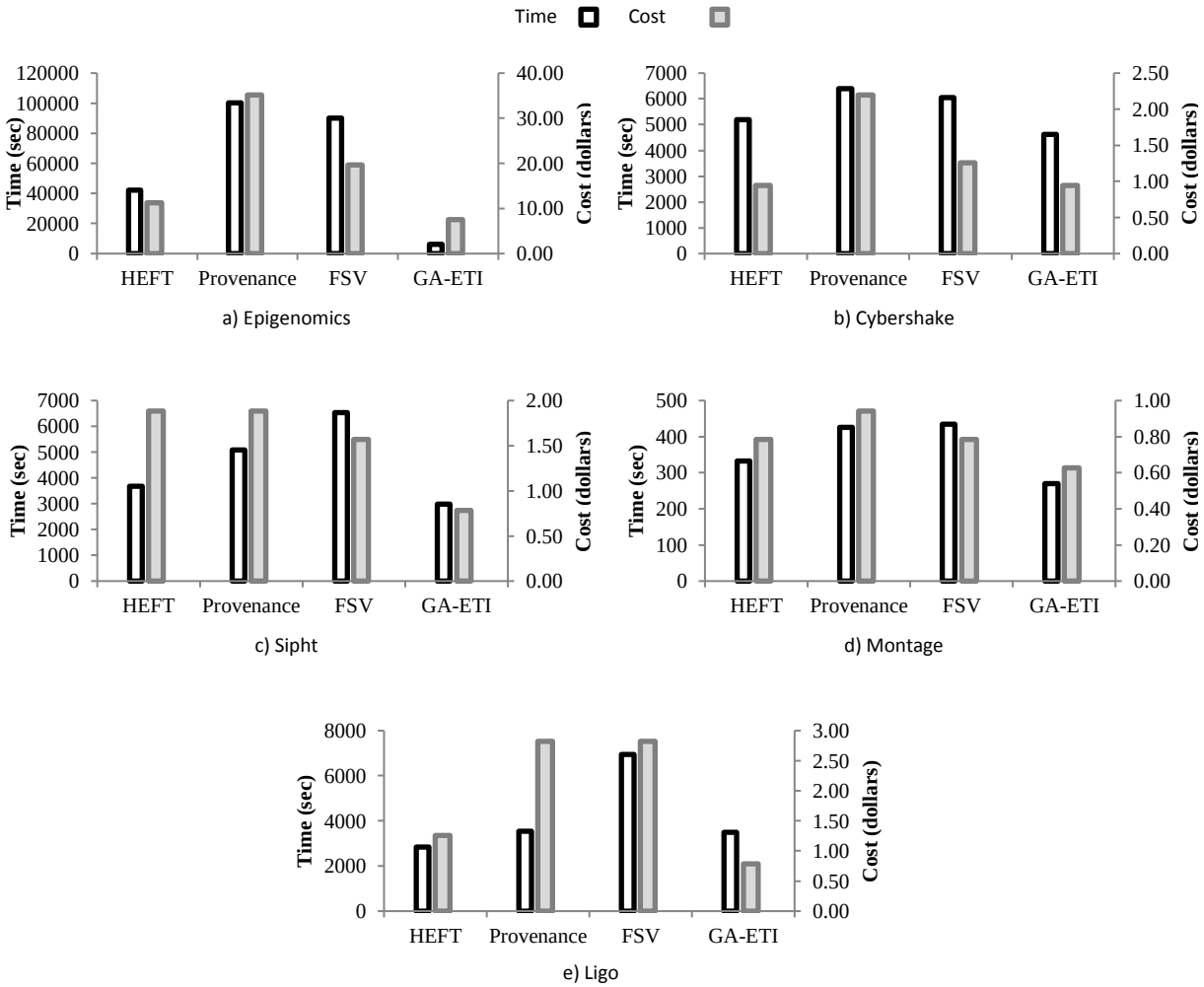


Figure 42. Best configuration execution time and monetary cost.

5.8 Discussion – GA-ETI Performance

To show the efficiency of this approach, it is analyzed from the following perspectives.

GA-ETI performance superiority

In this section, GA-ETI's behavior is analyzed and compared with the other algorithms. For this matter, all approaches are forced to produce scheduling plans for all

possible number of VMs. Schedulers start mapping for a single machine incrementing the number of VMs until the execution time does not improve. This criterion has been chosen since increasing the number of VMs beyond such a point only increments monetary cost. Figure 43 presents these results.

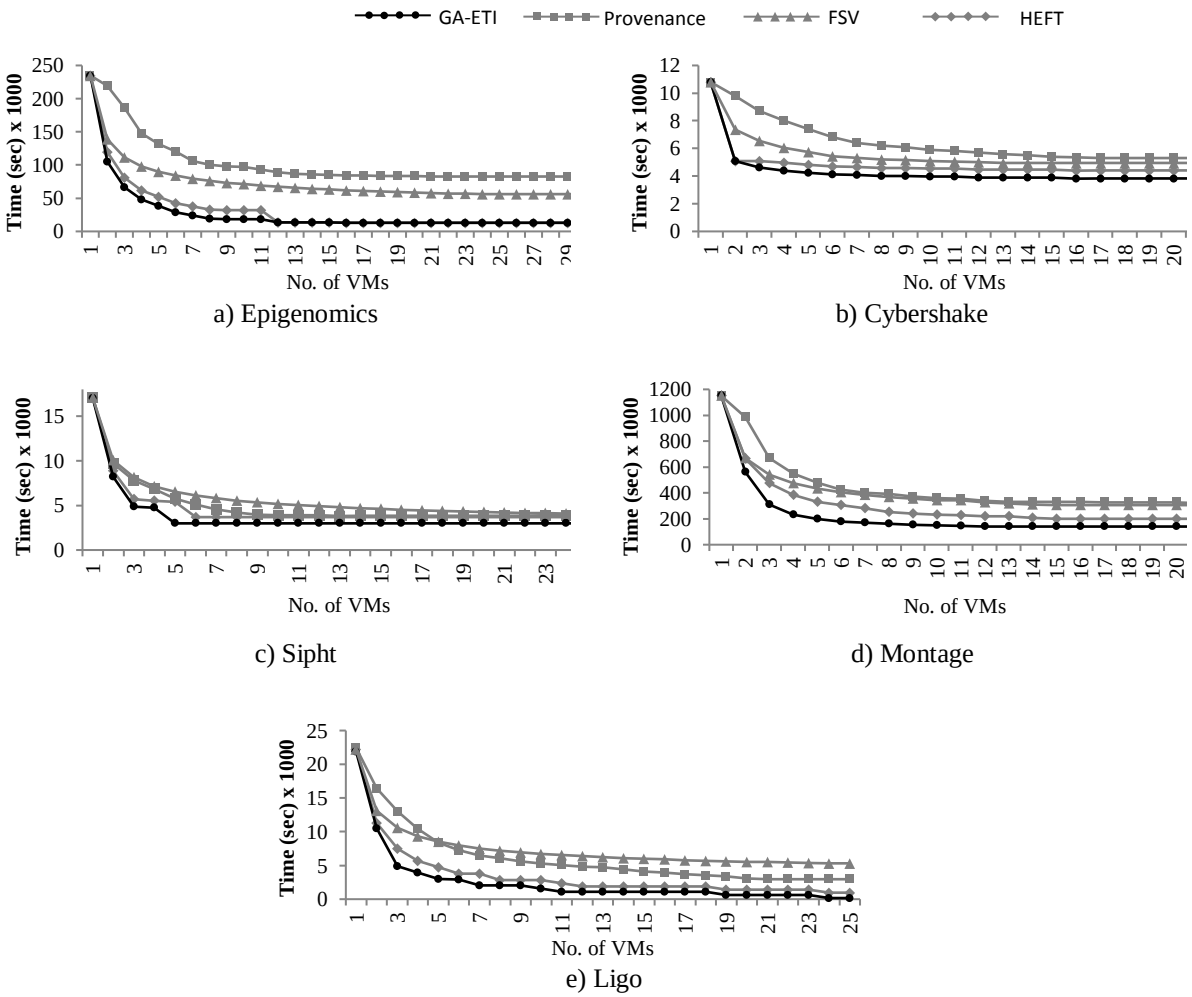


Figure 43. Execution time results with a different number of VMs.

Results from Figure 40 show that HEFT and GA-ETI present the lowest runtime execution time for the five workflows. Still, GA-ETI contributes the lowest values due to its scheduling policies. For instance, the Cybershake workflow presents particular dependencies where parallel nodes on the second level execute on 127.55 sec (63.35 sec for task execution plus ~64 sec for 791MB input file transfer on a 100Kbps network); internally, GA-ETI converged to solutions where groups of three of these parallel tasks are assigned to a single VM executing them serially in 254.05 sec transferring input files only once to the same VM.

In contrast, HEFT executes them parallel in 127.55 sec, transferring input set files to the different machines' VM. Overall, these decisions mean that HEFT requires redundant file transfers and executes the application in 5199 while GA-ETI only required 4619 seconds.

GA-ETI outperforms Provenance because the latter makes groups of tasks based on historical data. For example, the LIGO workflow on its second level has tasks that execute on ~400 sec setting grouping factor to be ~400. As a consequence on the following level, it tries to group as many tasks as possible to fulfill a total of ~400 sec, even though the next tasks execute on only ~5 sec causing the algorithm to group all tasks on the same VM. In contrast, GA-ETI provides flexibility to allocate tasks according to actual execution times. Finally, GA-ETI outperformed FSV because this latter executes workflows using Pegasus and HTCondor's default scheduling policies that are based on VM availability.

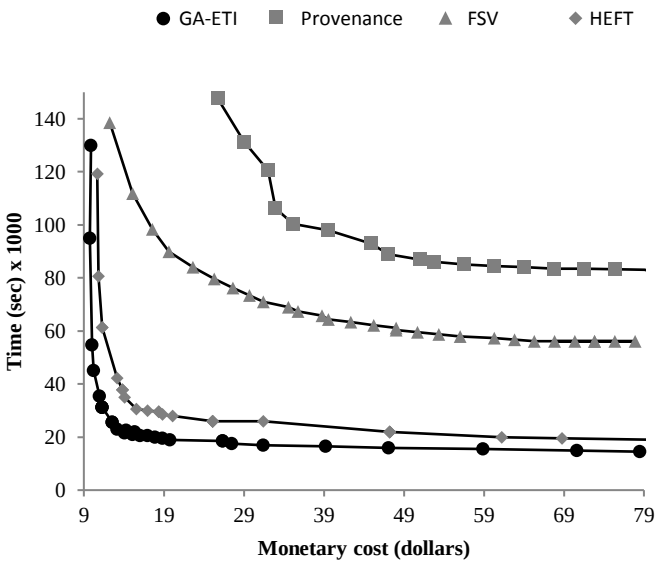


Figure 44. Epigenomics' $MSpn - MCst$ graph.

GA-ETI global search

This section presents results for makespan and monetary cost from the four schedulers in order to examine results distributed in the $MSpn\ vs\ MCst$ space. Figure 44 presents the $MSpn\ vs\ MCst$ graph for the Epigenomics workflow since the rest of the applications present similar behavior. On one hand, it is shown that Provenance and FSV present a semi-distribution of their results in the makespan-cost space. On the other hand, HEFT and GA-ETI present a stronger distribution of solutions along the space. This exercise proves that

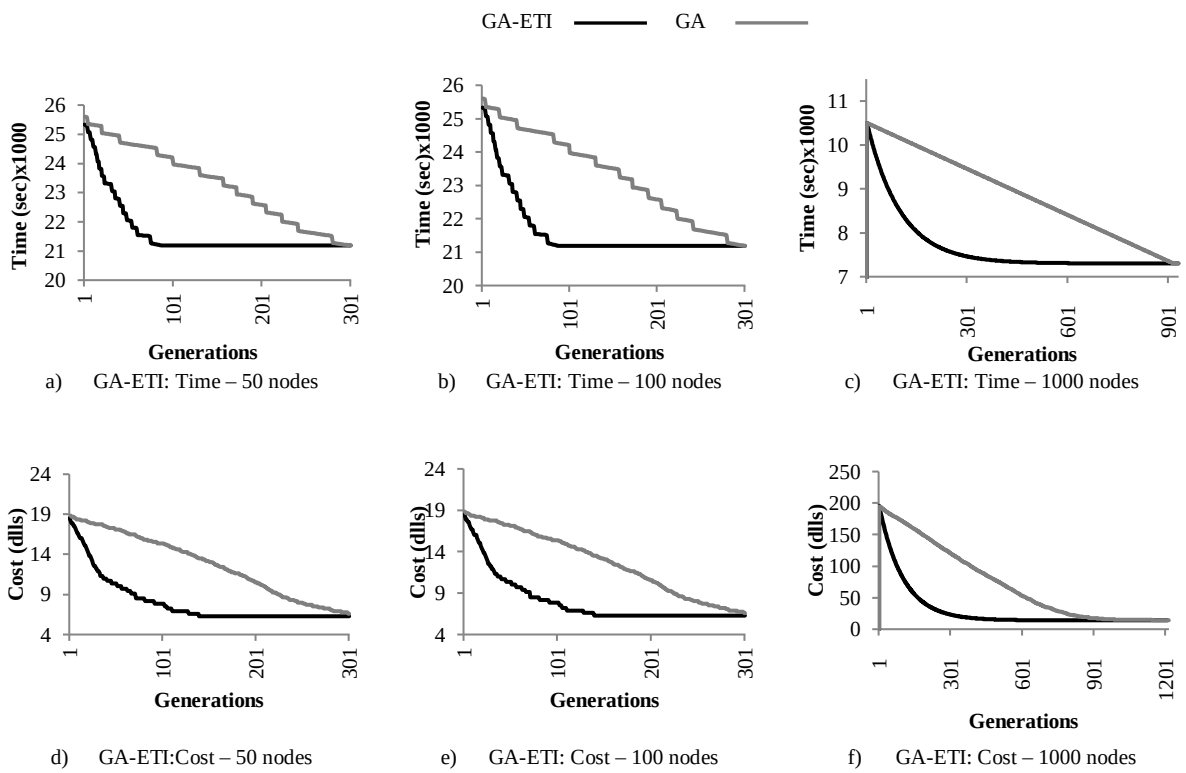
GA-ETI considers chromosomes distributed over the complete search space without being trapped at isolated locations. Additionally, GA-ETI's algorithm configuration allows it to consider solutions from the complete solution space without elite chromosomes driving it to specific regions.

GA-ETI behavior

In this last experiment section, GA-ETI was allowed to produce generations until no benefit was observed. Since workflows present similar behavior in terms of population evolution, this section only presents the results from a single workflow type. Additionally, three different workflow sizes were included for a deep analysis. For evaluation purposes, GA-ETI is compared against the general GA. Original GA uses conventional crossover and a swap mutation while GA-ETI additionally employs clustered crossover, increment and decrement mutation mechanisms.
Figure 45 presents results for population evolution on the Epigenomics workflow.

GA-ETI is able to converge to a satisfactory solution with fewer generations due to its enhanced crossover and mutation operators. These mechanisms complement each other, transforming the original GA into a potent tool to resolve the programming problem. On one side, clustered crossover avoids random selection of crossover points, instead, it first identifies workflow levels then it breaks chromosomes into clusters that later combine to produce offspring. This procedure allows the algorithm to combine the clusters of genes instead of chromosomes being randomly divided. On the other side, increment/decrement mutation provides an instrument to add/remove a particular VM from a chromosome allowing the algorithm to restructure that particular chromosome. The application of the mentioned operators allows GA-ETI to reduce randomness, an inherent characteristic from the original GA in converging to a final result.

A closer look at these graphs also reveals thought-provoking facts on execution time graphs. The difference between execution time obtained at the beginning and end of algorithms is minimal for both attacks. This is caused due to algorithms having access to an unlimited number of VMs allowing algorithms to take advantage of parallelism. This usually results in high monetary costs, for this reason the main challenge of algorithms is to allocate tasks to a reduced number of VMs while maintaining a low execution time.



a) GA-ETI: Time – 50 nodes
b) GA-ETI: Time – 100 nodes
c) GA-ETI: Time – 1000 nodes
d) GA-ETI:Cost – 50 nodes
e) GA-ETI: Cost – 100 nodes
f) GA-ETI: Cost – 1000 nodes

Figure 45. GA-ETI generations analysis for the epigenomics workflow.

5.9 Summary

This chapter presents GA-ETI, a scheduler for scientific applications for cloud systems to concurrently optimize their execution makespan and monetary cost. GA-ETI enhanced the original GA through purposeful/tailor-made modification to its crossover and mutation operators. GA-ETI uses enhanced crossover to combine clusters of genes rather than randomly divided chromosomes; it also employs increment/decrement mutations to add/remove virtual machines from a given chromosome. Both modifications yield reduced inherent randomness compared to the original GA. Using five workflows to represent a variety of current scientific problems, GA-ETI was tested and proved its superiority against three (HEFT, Provenance and FSV) well-known/up-to-date schedulers in this field. GA-ETI solutions had lower makespan and monetary cost when compared with solutions provided by HEFT. Unlike FSV, GA-ETI produces a complete scheduling configuration prior to execution with better qualities. In contrast to Provenance, GA-ETI produces its own scheduling configuration and uses a workflow manager system only as middleware to execute scheduling decisions. GA-ETI also revealed that, despite the general impression, optimal

execution of workflows does not require a high number of resources (compared to the number of parallel nodes) in most cases.

6 PSO-DS: A Scheduling Engine for Scientific Workflow Managers

6.1 Preliminaries

This chapter presents PSO-DS (Particle Swarm Optimization with Discrete adaptation and a featured SuperBEST), an algorithm to solve the problem to schedule large workflows. PSO-DS employs the distribution mechanisms of BaRRS and readapts the GA-ETI chromosomes into particles swarming for an optimal position.

Firstly, Section 6.2 presents the required definitions to understand the cloud model to run the PSO-DS. Based on this model, Section 6.3 formally expresses the problem to solve. Then, Section 6.4 presents the PSO-DS. In order to validate the PSO-DS, Section 6.5 describes the experiment setup. Then, Section 6.6 present results. Section 6.7 analyzes the scheduling of large size workflows. Next, Section 6.8 discusses algorithm performance. Finally, Section 6.9 provides a summary with the remarkable features of experiments.

The main contributions of the PSO-DS are: (1) an adapted PSO scheduler capable of managing large workflow sizes in a record time, (2) a low complexity scheduling algorithm

based on the particle swarm optimization with upstanding results in terms of execution time and monetary cost, and (3) presents a novel particle scheduling reconstruction, and introduces a Super Particle that shorts the PSO search time for an optimal result. The core of this chapter has been sent for publication to the *Journal of Supercomputing* [3].

6.2 Framework

Figure 46 presents the framework that the PSO-DS employs to execute workflows on cloud environments. In this architecture: (1) a user develops his/her workflow $\boldsymbol{W} = \{t_1, \dots, t_n\}$ to solve a problem in a particular scientific area. Each task t_i from $\boldsymbol{W}$ has a set of parents $t_i^{parents}$ linked by a set of files with total size f^{size}. Then on (2), framework refers to execution time $\hat{t}_i^{exe}$ for each task. This framework recommends that a user collects the calculated execution time from previous executions and then attaches it to each task. The user selects the optimization level for each objective, i.e. monetary cost and makespan. It is important to highlight that the user selects a percentage and not a budget or time value limit since PSO-DS provides an analysis and then provides different options based on user selection. Then at stage (3) the PSO-DS receives workflow with its information attached including optimization levels. The user then receives feedback with the possible scheduling configuration with different number of VMs, monetary cost $MCst$ and makespan $MSpn$ then he/she selects the one best suited to his situation. Finally, at stages (4-5), the cloud system provides the required VMs and the scheduler submits tasks to each resource based on the selected scheduling configuration and triggers execution.

6.3 The Scheduling Problem

The scheduler's responsibility is to organize tasks into a set of vm_j^{queue} to be executed by a given vm_j with the objective of minimizing total makespan $MSpn$ (Eq. 21) and monetary cost $MCst$ (Eq. 22). To accomplish this task, the scheduler is required to define the size of the pool of resources $\boldsymbol{VM} = \{vm_1, \dots, vm_v\}$ based on the cost to hire each resource vm_j^{cost} and the potential time to execute each set of tasks vm_j^{time} from a given vm_j. Makespan is the Latest Finishing Time (LFT) to execute all vm_j^{queue}. Eq. 23 defines vm_j^{time} as the LFT from all tasks assigned to vm_j. Eq. 24 expresses $\hat{t}_i^{total}$ as the time to execute t_i while Eq. 25 is the time to execute all its parent tasks. Finally, Eq. 26 is employed

to calculate the file transfer between vm_p and vm_i. Figure 47 presents an example to calculate $MSpn$ and $MCst$ for a four task workflow and a two VM set with unitary values for $\hat{t}_i^{exe}$, f_i^{size}, vm_j^{bw} and vm_j^{cost}. Firstly, consider that the four tasks are equally distributed to the set of VMs as indicated. Secondly, files are transferred only between tasks residing on different VMs. Thirdly, tasks are presented over a timeline exhibiting its corresponding execution time $\hat{t}_i^{exe}$ and its transfer time $\hat{t}_i^{f}$. Task t_1 requires transfer of its corresponding f_1^{size}while t_2 does not required data transmission since it is allocated to the same VM. In contrast, t_3 requires to transfer f_3^{size} from vm_1 to vm_2 while t_4 does not required any data transmission. It is clearly seen that vm_1^{time} executes its set of tasks within three units of time while vm_2^{time} executes its tasks in five units of time. Finally, $MSpn$ selects the largest value from $[vm_1^{time}, vm_2^{time}]$ obtaining the value of five. As for $MCst$, it obtains a value of eight units of time.

$$MSpn = LFT_{j=1}^{|VM|}\,[vm_j^{time}] \qquad \text{Eq. 21}$$

$$MCst = \sum_{j=1}^{|VM|} \lceil vm_j^{time} \rceil vm_j^{cost} \qquad \text{Eq. 22}$$

$$vm_j^{time} = LFT_{i=1}^{\left|vm_j^{queue}\right|}\left[\,\hat{t}_i^{total}\right] \qquad \text{Eq. 23}$$

$$\hat{t}_i^{total} = \hat{t}_i^{f} + \hat{t}_i^{exe} + \hat{t}_i^{parent} \qquad \text{Eq. 24}$$

$$\hat{t}_i^{parent} = \sum_{p=1}^{|parents|} \hat{t}_p^{f} + \hat{t}_p^{exe} \qquad \text{Eq. 25}$$

$$\hat{t}_i^{f} = \frac{f_i^{size}}{min\left(vm_p^{bw}, vm_i^{bw}\right)} \qquad \text{Eq. 26}$$

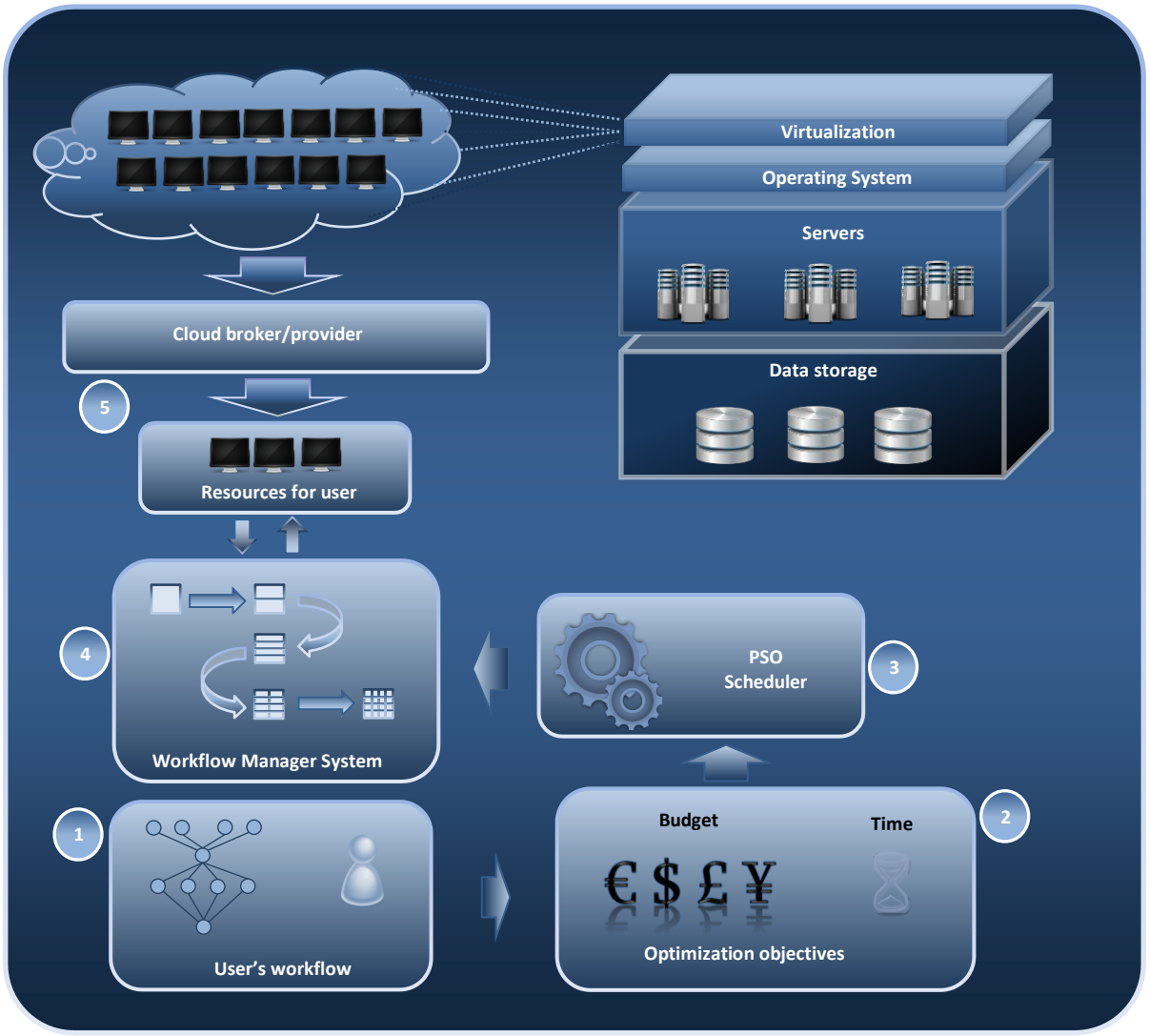


Figure 46. Cloud customer-provider affiliation.

In this model it is assumed that every VM has a fixed bandwidth (vm_j^{bw}), number of cores (vm_j^{cores}), memory size (vm_j^{mem}) and disk size (vm_j^{disk}). Since this study uses Pegasus as part of the experimentation, workflow description follows its format including executable files, input data and DAX file which is an abstract description of the workflow and its internal dependencies. Additionally, workflow must specify the time to execute each of its task. Once information is complete, the analyzer produces scheduling plans with different finishing time, monetary cost and number of VMs. For this task PSO optimization techniques are employed as described in next section.

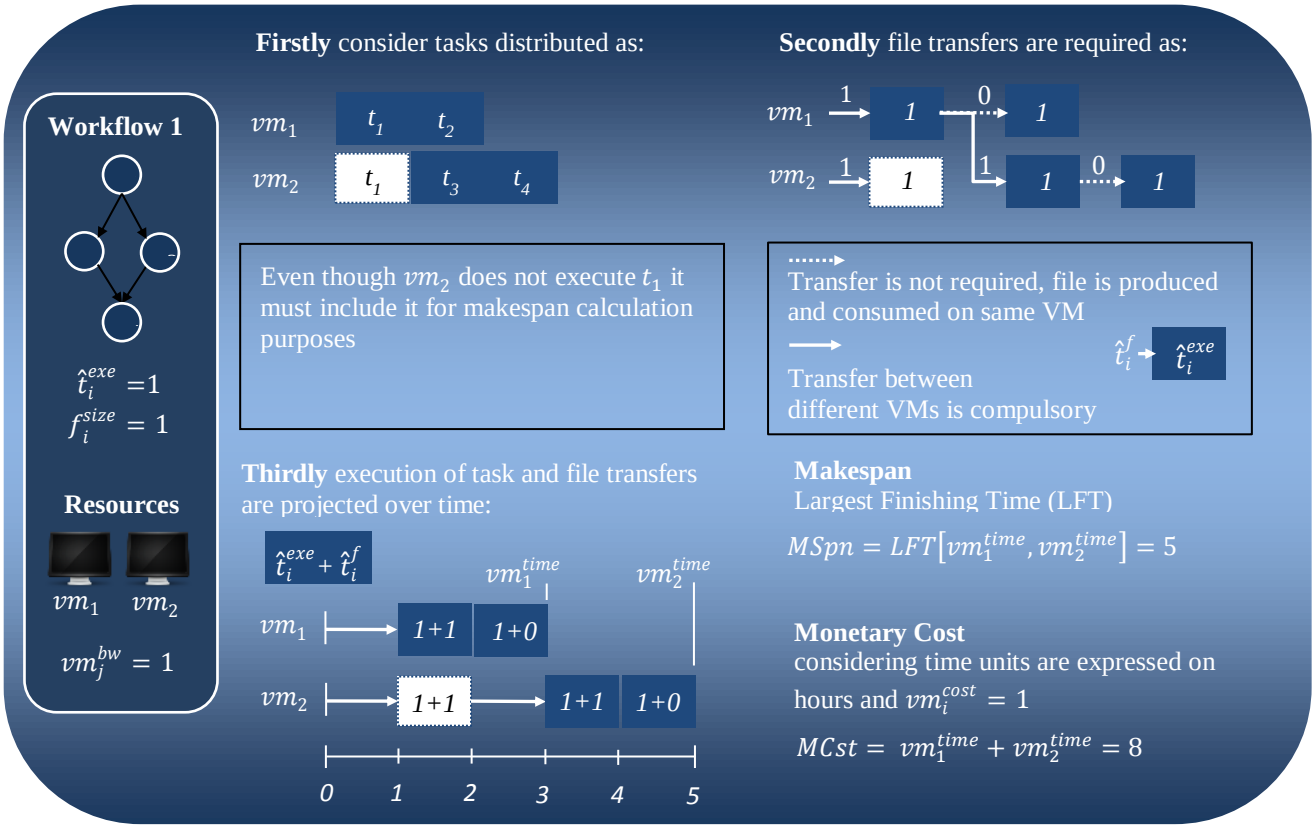


Figure 47. Example to calculate makespan and monetary cost.

6.4 PSO-DS Approach

For this study, the author developed a scheduling approach based on the Particle Swarm Optimization (PSO) mechanism to solve the aforementioned problem. The PSO is a process to find a solution for nonlinear problems in terms of their optimization functions. PSO is based on particles continuously moving while aiming to obtain the coordinates that optimize the evaluation value as illustrated in Figure 48. The PSO is strongly related to swarming theory and has similarities with genetic algorithms (GAs) [51]. As compared with GA, PSO has lower demands in terms of computational power, memory capacity, and computer coding with exceptional capabilities to solve different kinds of optimization problems [54].

6.4.1 Particle Swarm Optimization (PSO)

In the original form of PSO [51], the GBEST model is a searching technique in the solution space for an optimal answer. It is orientated for problems expressed with real

numbers. In order to extend the PSO scope, a discrete version of the swarm algorithm is developed in [52]. The core of the original version was kept intact, while differing only on the discrete mode to manipulate the problem. Each particle in the PSO represents a solution, has a position and a velocity in the search space. Through a series of iterations, particles *swarm* through the solution space to find the maximum (or minimum) value for a given evaluation function. The following is the notation to introduce the discrete PSO (Algorithm 1).

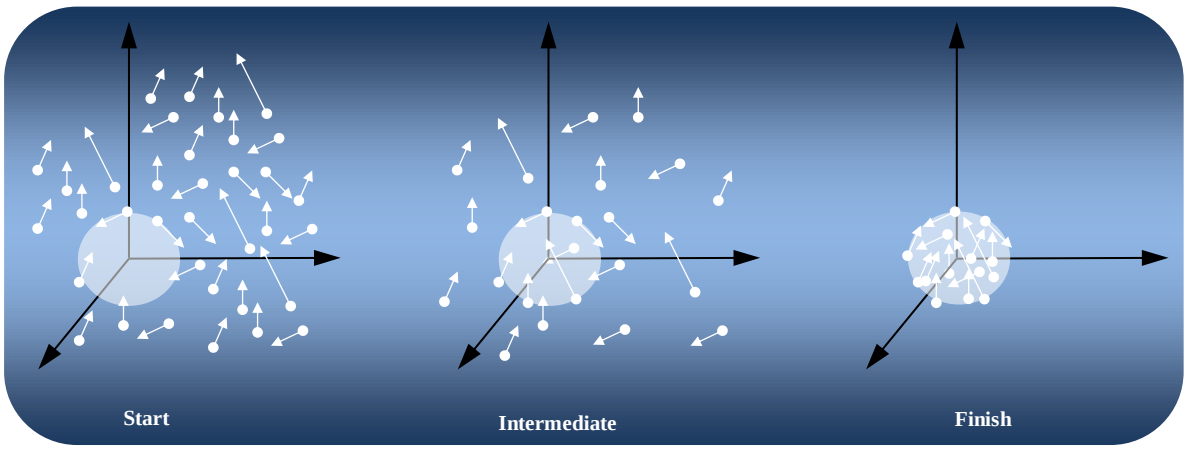


Figure 48. The PSO process.

On a population with size P, consider the position of the *i-th* particle as $X_i^t = (x_{i1}^t, x_{i2}^t, \dots, x_{iD}^t)$ with D bits where $x_{id}^t \in \{0, 1\}$. The particle's velocity is then defined as $V_i^t = (v_{i1}^t, v_{i2}^t, \dots, v_{iD}^t)$ where $v_{id}^t \in R$. The PSO keeps a record of particle's best position on $PBEST_i^t = (pbest_{i1}^t, pbest_{i2}^t, \dots, pbest_{iD}^t)$ as well as a global best solution ever found in $GBEST^t = (gbest_{i1}^t, gbest_{i2}^t, \dots, gbest_{iD}^t)$. Eq. 27 presents the function to calculate the velocity v_{id}^t for the *d-th* dimension of the *i-th* particle on the t iteration. The ω term introduced in [76] is a particle's inertia to continue moving towards its original direction. Acceleration coefficients of c_1, c_2 act as the particles' memory, inclining it to move toward $PBEST_i^t$ and $GBEST^t$, respectively. The objective of v_{id}^t is to drive a particle in the direction of a "superior" position in terms of its evaluation function value. Position X_i^t is updated on every iteration of the PSO.

Sigmoid function (Eq. 28) is employed to operate velocities, as probabilities values, in the interval of [0, 1]. Additionally, v_{id}^t is limited to a fixed range of values $[-V_{max}, +V_{max}]$

to prevent $s(v_{id}^t)$ from falling on the upper or lower bound of [0, 1]. In this study's experiments, also advised by [77], $V_{max} = 4$. Algorithm 1 presents the generic discrete version of the PSO for a maximization optimization. In step 1, it initializes an array of particles with random positions X_i^0 and velocities V_i^0. In steps 2-20, it executes its main cycle. It employs function F to evaluate each particle's value in step 4, if a particle's value is greater than its previous best position then $PBEST_i^t$ is updated by the particle's value. Similarly, the global best $GBEST^t$ value is compared, and updated if required, with P_i^t (steps 7-9). In the sub-cycles in steps 10-18, Algorithm 1 updates the velocities and positions for all D dimensions of a particle. In steps 11-12, each particle updates its velocity v_{id}^t and caps its values. Finally, based on the result of the sigmoid function, each particle sets the value for each dimension x_{id}^{t+1} for iteration $t + 1$. The main cycle continues until a termination criterion is met.

Algorithm 1 Discrete PSO – The Global model

F: Evaluation Function

```
1:  Initialize an arrangement of P particles
2:  While a termination criterion is not met
3:      For each X_i^t particle in N_p
4:          if F(X_i^t) > F(PBEST_i^t)
5:              PBEST_i^t = X_i^t
6:          end
7:          if F(P_i^t) > F(GBEST)
8:              GBEST = P_i^t
9:          end
10:         For each dimension d in D
11:             Update v_id^t (Eq. 27)
12:         Limit v_id^t ∈ [−V_max, +V_max]
13:             if s(v_id^t) > random [0,1]
14:                 x_id^{t+1} = 1
15:             else
16:                 x_id^{t+1} = 0
17:             end
18:         end
19:     end
20: end
```

$$v_{id}^t = \omega v_{id}^{t-1} + c_1 r_1 (pbest_{id}^t - x_{id}^t) + \quad c_2 r_2 (gbest_{id}^t - x_{id}^t) \qquad \text{Eq. 27}$$

$$s(v_{id}^t) = \frac{1}{1 + \exp(-v_{id}^t)} \quad \text{Eq. 28}$$

6.4.2 The PSO-DS

This section presents the modified Particle Swarm Optimization with Discrete adaptation and a featured *SuperBEST* (PSO-DS) –an extension to the generic PSO– to solve the scheduling problem in this article. Following is the description of the PSO-DS particles, velocity and the introduction of a featured *SuperBEST* particle.

Adaptation of particle format

Similarly to [55], PSO-DS requires unfolding of the original discrete PSO particles to interpret integer numbers to solve the scheduling problem. In PSO-DS, particles have an augmented format $X_i^t = (x_{i11}^t, x_{i21}^t, \dots, x_{inv}^t)$, $x_{ijk}^t \in \{0,1\}$, where $x_{ijk}^t = 1$ if the *j-th* task of the *i-th* particle is executed in vm_k, and $x_{ijk}^t = 0$ otherwise. For ease of explanation, this study introduces a short format to represent particles i.e. $X'^t_i = (x'^t_{i1}, x'^t_{i2}, \dots, x'^t_{in})$, $x'^t_{ij} \in \{vm_1, \dots, vm_v\}$, is the abstract representation of particle X_i^t where x'^t_{ij} is the vm_k executing task j of particle i at time t. An example particle is presented in Figure 49 for a workflow with four tasks $\boldsymbol{W} = \{t_1, t_2, t_3, t_4\}$, and a set of two resources $\boldsymbol{VM} = \{vm_1, vm_2\}$. Here, particle i is expressed in its long and abstract format X_i^t and X'^t_i, respectively.

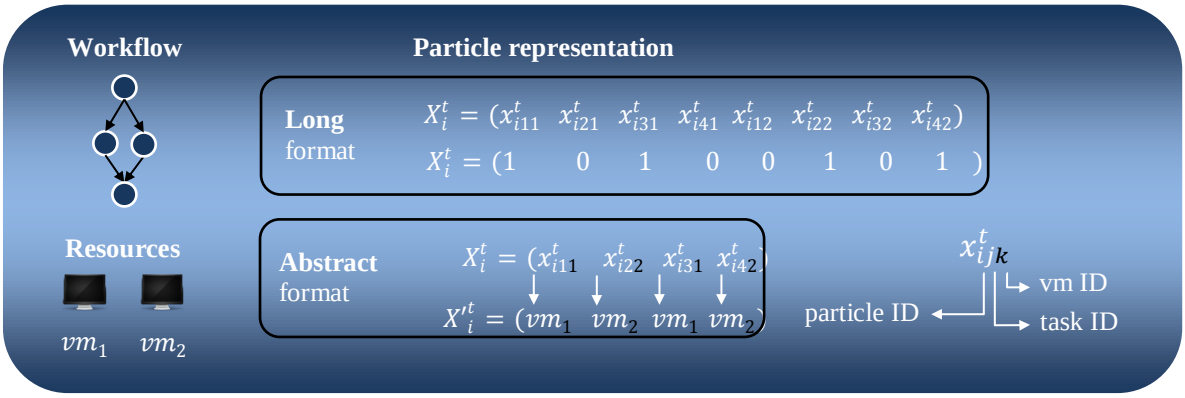


Figure 49. Particle representation for the PSO-DS.

For a particle expressed in its abstract format X'^{t}_{i}, the number of different values that each of its dimensions can have is v, given $\boldsymbol{VM} = \{vm_1, \ldots, vm_v\}$. At the same time, v is driven by the number of parallel tasks in a given workflow. Figure 50 illustrates this concept; for Workflow 1, the maximum number of tasks that can be executed in parallel is four (in the third level of Workflow 1). As a consequence, $\boldsymbol{VM}$ is set to the values of $\{vm_1, vm_2, vm_3, vm_4\}$ because any additional VM (more than four) in the pool will remain idle during execution of this workflow.

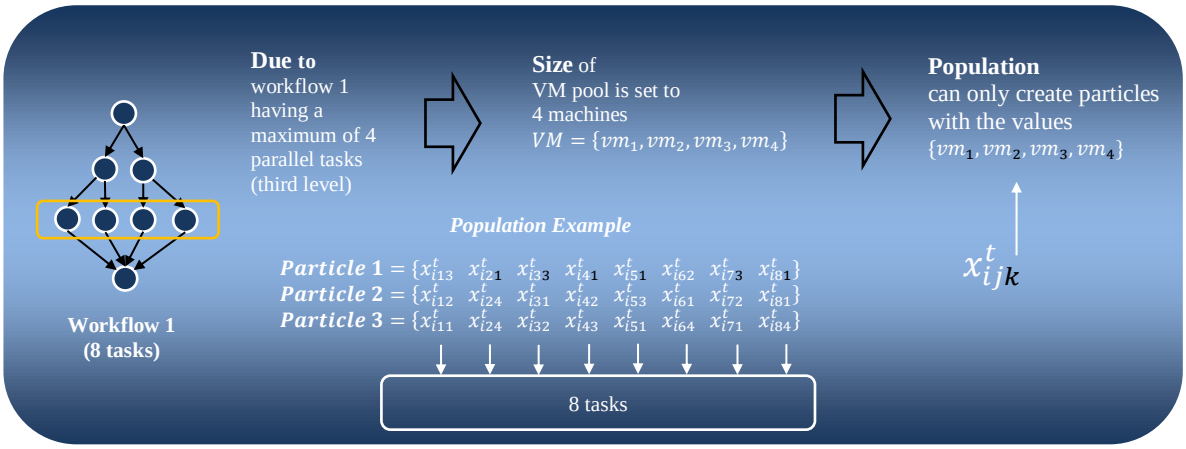


Figure 50. PSO-DS particle example.

Particle's velocity adaptation

The velocity from Eq. 27 is transformed into Eq. 29 in order to follow each particle's adaptation. Firstly, for X^t_i (expressed in its long format), the best position is defined as $PBEST^t_i = (pbest^t_{i11}, pbest^t_{i21}, \ldots, pbest^t_{inv})$, while the global best particle in the population is defined as $GBEST^t = (gbest^t_{11}, gbest^t_{21}, \ldots, gbest^t_{nv})$. Parameters ω, c_1, r_1, c_1, r_1 and V_{max} have the same functions as in the original discrete PSO described in the previous section. Additionally, Eq. 28 is slightly modified to produce Eq. 30 for managing dimension velocities as a set of probabilities; i.e. representing velocities in the range of [0,1]. Here, each dimension's velocity v^t_{ijk} is the probability of vm_k to execute t_j.

$$v^t_{ijk} = \omega v^{t-1}_{ijk} + c_1 r_1 \left(pbest^t_{ijk} - x^t_{ijk} \right) + \quad c_2 r_2 \left(gbest^t_{jk} - x^t_{ijk} \right) \qquad \text{Eq. 29}$$

$$s(v_{ijk}^t) = \frac{1}{1 + \exp(-v_{ijk}^t)} \quad \text{Eq. 30}$$

6.4.3 Particle reconstruction

PSO-DS has a population of solutions expressed as velocity probabilities requiring an interpretation to construct scheduling configurations. In this context, VMs compete to execute tasks, while each task can only be assigned to a single resource. In contrast to [55] where construction of particles forces consecutive tasks to be assigned to different VMs, our approach allows assignment of sets of successive tasks to the same VM to avoid unnecessary data transfers. Consider $s'(v_{ijk}^t)$ (Eq. 31) as the probability of assigning the j-th task to the k-th resource from the pool of v machines where $\sum_{k=1}^{v} s'(v_{ijk}^t) = 1$. During a process to be repeated for every task j in every particle i, only one dimension, namely k-th, $x_{ijk}^t = 1$, while $\{x_{ijk'}^t = 0 \mid k' \neq k\}$; k is the index of the dimension with the maximum value. Figure 51 presents a velocity and position update example for a single task workflow and a set of four VMs. Here, the $GBEST^t$ indicates allocating the task to vm_4, while the particle X_i^t has assigned the task to vm_1. In the resulting $S'(V_i^t)$, the last dimension exhibits the highest probability to obtain 1, and thus in the updated particle X_i^{t+1}, vm_4 executes t_1.

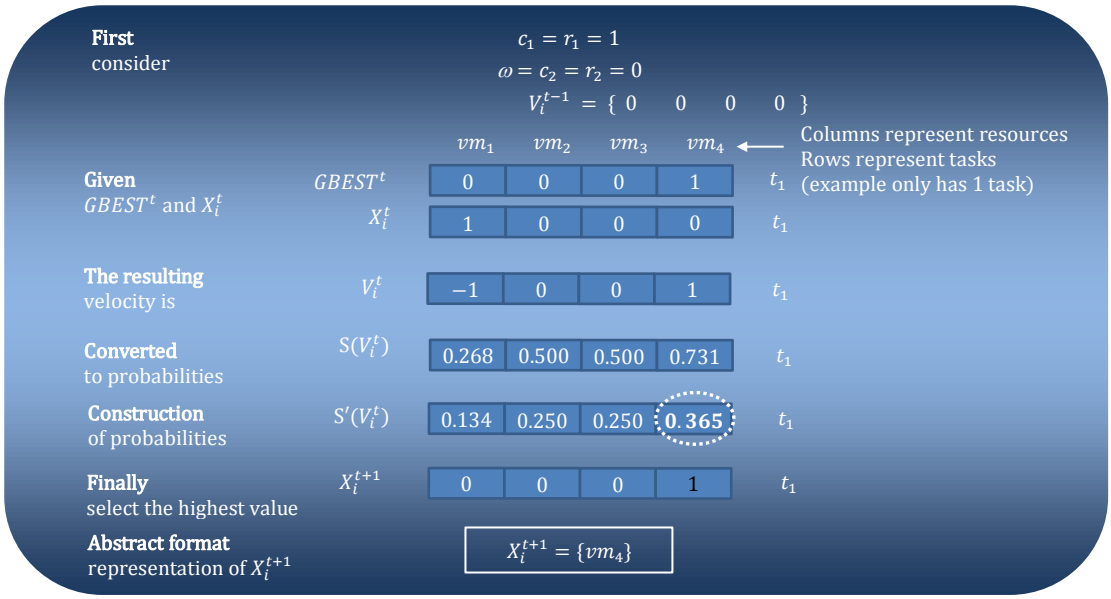


Figure 51. Particle velocity update example.

$$s'(v_{ijk}^t) = \frac{s(v_{ijk}^t)}{\sum_{k=1}^{v} s(v_{ijk}^t)} \quad \text{Eq. 31}$$

6.4.4 SuperBEST Particle and the GBEST

A new particle was defined, namely $SuperBEST^t$, in PSO-DS; $SuperBEST^t$ is built using the most popular particles' elements in the population. Consider the set $X''^t_j = \{x''^t_{j1}, x''^t_{j2}, \dots, x''^t_{jP}\}$, $x''^t_{ji} \in \{vm_1, \dots, vm_v\}$, where x''^t_{ji} is the value assigned to dimension j in the $i-th$ particle (expressed in its abstract format) for a population with P particles. In this case, $SuperBEST^t = (sb^t_1, sb^t_2, \dots, sb^t_n)$, where sb^t_j holds the value with highest frequency from X''^t_j. Figure 52 illustrates the formation of the $SuperBEST^t$. Firstly, from a population of three particles X^t_1, X^t_2 and X^t_3, a set of X''^t_j vectors expresses the population; then the frequency of occurrence for each vm_k is counted; and finally, the $SuperBEST^t$ is composed. The PSO-DS uses the $SuperBEST^t$ to build the $GBEST^t$. For $GBEST^t_{Pbased} = max^P_{i=1}[PBEST^t_i]$ and $GBEST^t_{Xbased} = max^P_{i=1}[XBEST^t_i]$, the $GBEST^t = max(SuperBEST^t, GBEST^t_{Pbased}, GBEST^t_{Xbased})$.

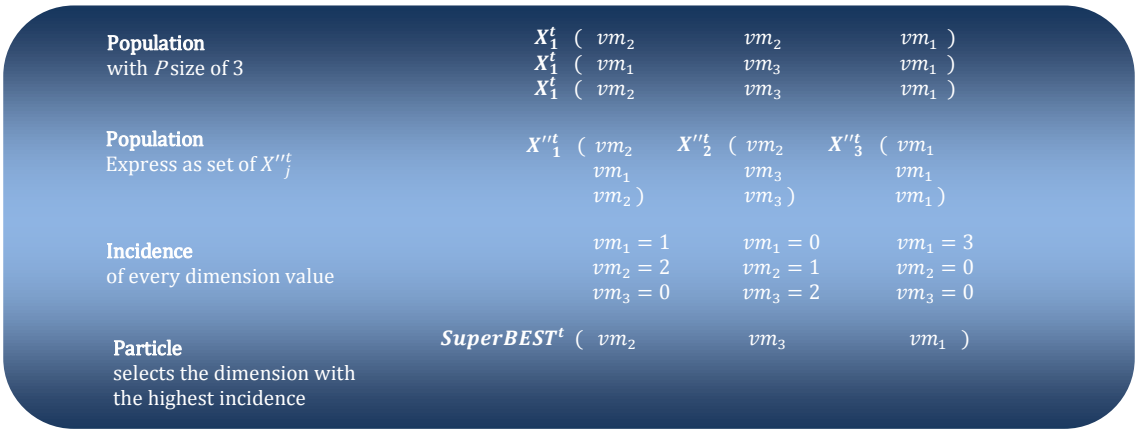


Figure 52. The *SuperBEST* particle formation.

Evaluation function and termination criterion

This study adopts a maximization optimization for the scheduling process in the PSO-DS. It employs an evaluation function E_{value} that integrates the makespan $MSpn$ and monetary cost $MCst$ with weight values w_1 and w_2 to control objective optimization. The variables max^{time}, min^{time}, max^{cost} and min^{cost} retain maximum and minimum values of makespan and economical cost through continuous update during PSO-DS processes. PSO-DS continues until E_{value} shows no improvement; PSO-DS outputs GBEST as the final solution.

$$E_{value} = w_1 \frac{(max^{MSpn} - MSpn)}{(max^{MSpn} - min^{MSpn})} + w_2 \frac{(max^{Cst} - MCst)}{(max^{Cst} - min^{Cst})} \quad \text{Eq. 32}$$

Algorithm 2 PSO-DS

$\boldsymbol{E_{value}}$ Evaluation Function, Population size P, V_{max}, Workflow $\boldsymbol{W}$

Initialize an arrangement of P particles with random positions and
While a termination criterion is not met
  For all particles ($i = 1\ to\ P$)
    if $\boldsymbol{E_{value}}(X_i^t) > \boldsymbol{E_{value}}(PBEST_i^t)$
      $PBEST_i^t = X_i^t$
    end
    if $\boldsymbol{E_{value}}(X_i^t) > \boldsymbol{E_{value}}(GBEST^t)$
      $GBEST_{Xbased}^t = XBEST_i^t$
    end
    if $\boldsymbol{E_{value}}(PBEST_i^t) > \boldsymbol{E_{value}}(GBEST^t)$
      $GBEST_{Pbased}^t = PBEST_i^t$
    end
  end
  For all tasks ($j = 1\ to\ n$)
    $X''^t_j = \{x''^t_{j1}, x''^t_{j2}, \dots, x''^t_{jP}\}$
    $sb_j^t = higher\_incidence(X''^t_j)$
    $SuperBEST^t \leftarrow sb_j^t$
  end
  $GBEST^t = \max(SuperBEST^t, GBEST_{Pbased}^t, GBEST_{Xbased}^t)$
  For all particles ($i = 1\ to\ P$)
    For all tasks ($j = 1\ to\ n$)
      For all VMs ($k = 1\ to\ v$)
        $v_{ijk}^t = \omega v_{ijk}^{t-1} + c_1 r_1 \left(pbest_{ijk}^t - x_{ijk}^t \right) +$
        $v_{ijk}^t \in [-V_{max}, +V_{max}]$
        $s\left(v_{ijk}^t\right) = 1/\left(1 + \exp(-v_{ijk}^t)\right)$
        $x_{ijk}^t = 0$
      end
      $s'\left(v_{ij(0)}^t\right) = 0$
      For all VMs ($k = 1\ to\ v$)
        $s'\left(v_{ijk}^t\right) = s\left(v_{ijk}^t\right) / \sum_{k=1}^{v} s\left(v_{ijk}^t\right)$
        if $s'\left(v_{ijk}^t\right) > s'\left(v_{ij(k-1)}^t\right)$
          Save k
        end
      end
      $x_{ijk}^t = 1$
    end
  end
end

6.4.5 PSO-DS Algorithm

The resulting PSO-DS algorithm is presented in Algorithm 2. In step 1, it creates a population with size P where random positions X_i^t and velocities V_i^t are assigned to the population. Then steps 2 – 38 present the main loop. Steps 4 – 6 update $PBEST_i^t$ for every particle; $GBEST_{Xbased}^t$ and $GBEST_{Pbased}^t$ update their values if the current particle has a higher evaluation value. In steps 14 – 18, the algorithm builds the $SuperBEST^t$ particle. Following in step 19, PSO-DS selects $GBEST^t$ from the pool $\{SuperBEST^t, GBEST_{Pbased}^t, GBEST_{Xbased}^t\}$. In steps 23 – 26, velocity $v_{ijk}^t$is updated and capped to the range $[-V_{max}, +V_{max}]$. Next, s(v_{ijk}^t) expresses v_{ijk}^t as a probability in the interval [0,1]; its respective position x_{ijk}^t is set to 0 as a preliminary step to update each particle's position. Steps 28 – 35 present procedures to update each particle's position; it is repeated n times for a given workflow $\boldsymbol{W}$ with size n. Steps 29 – 34 form a loop to calculate $s'(v_{ijk}^t)$ for all k resources that are competing to execute the j-th task in the i-th particle. Finally, only the resulting k resource (Step 32) in x_{ijk}^t obtains the value of 1 for the j-th task in the i-th particle.

6.5 Experiment Setup

This section evaluates the performance of the PSO-DS using three main tests. Experiment 1tested and compared the performance of the PSO-DS and the Pegasus-WMS to schedule large workflows. Then experiment 2, analyzed the need to guide the user in selecting a limited budget by comparing monetary costs when executing large workflows with/without an unlimited budget. Finally experiment 3 compared the performance of the PSO-DS against Provenance [72], HEFT [13], FSV [7] and GA-ETI [2] in scheduling large workflow size.

The experimental test bed consists of a cloud computing system with three Krypton Quattro R6010 with 4-way AMD OpteronTM 6300 series (64-Cores each). The Pegasus-WMS (4.2) was selected as the WMS, it is installed on Ubuntu 14.04 operating system. The VMware vSphere (5.5) manages computer resources and provides virtual machines with the aforementioned platform. The PSO-DS performs as the scheduling engine with parameters set as shown in Table 8. Five scientific workflows were selected from [4] to produce the experiments. Workflows represent applications from different scientific areas including

astronomy, geology, biology, cosmic analysis and biotechnology. Their details are presented in Table 9 (see Subsection 4.2

Table 8. PSO-DS setup

Parameter	Symbol	Value
Population	P	100
Makespan optimization	w_1	0.5
Monetary cost optimization	w_2	0.5
Social acceleration coefficient	c_2	2
Personal acceleration coefficient	c_1	2
Velocity limit	V_{max}	4
Inertia coefficient	ω	1.2

Table 9. Characteristics of the scientific workflows employed in experiments to test GA-ETI

	Nodes	*w-levels*	*parallel-tasks*	Average file size (MB)	Average task execution time (s)	Dependencies patterns
Epigenomics	997	8	245	749	2346	(2)(3)(4)
Montage	1000	9	662	20.6	11.34	(2)(3)(4)
Cybershake	1000	5	494	1156.1	51.70	(1)(3)(4)
Ligo	1000	8	480	55.6	222.0	(1)(4)(5)
Sipht	1000	7	584	22.02	210.27	(4)(5)

(1) Process; (2) Pipeline; (3) Data distribution; (4) Data aggregation; (5) Data Redistribution

6.6 Results

This first experiment stage has the objective of highlighting the need to add a specialized scheduling analysis on top of WMS. Table 10 provides makespan and monetary cost results for the PSO-DS and Pegasus-WMS. Epigenomics workflow presents the biggest difference in terms of time and cost due to its great parallelism level. In this application, PSO-DS converges to solutions where dependent tasks are executed on the same VM; as a consequence PSO-DS successfully decreases the number of data transfers. Similar scenarios are presented on the rest of the applications. For example, the Ligo workflow, which has three main groups of tasks running in parallel, PSO-DS converges to solutions where not all of the tasks are executed at the same time, since monetary cost is also considered as an optimization objective it balances the optimization of makespan and cost. As for Pegasus concerns, it presents higher makespan and cost values as its only objective is to execute the application. Pegasus uses HTCondor as its internal DAG (Direct Acyclic Graph) executer. HTCondor receives workflow and sends its tasks for execution with almost no control on which tasks to execute on a given VM or which data files would be replicated. In contrast,

PSO-DS is able to evaluate a different number of scheduling configurations and chooses the one that contributes to the highest optimization value.

Table 10. Results for Makespan and monetary cost for PSO-DS and PEGASUS-WMS.

	Makespan		Monetary cost	
	PSO-DS	**Pegasus-WMS**	**PSO-DS**	**Pegasus-WMS**
Epigenomics	**180889**	809209	**40.035**	176.625
Cybershake	**39396**	49037	**6.908**	13.188
SIPHT	**26879**	64286	**6.28**	11.304
Montage	**1440**	3464	**0.785**	1.099
Ligo	**17094**	91572	**5.495**	20.41

The need to guide users in selecting a budget limit

This experiment allows the PSO-DS to produce scheduling configurations relaxing the monetary cost optimization. Results are presented in Figure 53, it first schedules for two VMs, then three, four and so on. For each workflow, three graphs are presented; the first graph presents evaluation function values, second and third graphs present their corresponding makespan and monetary cost. For each case, a red shadow highlights the values with function values above 80%. As readers will notice, makespan drops dramatically as the number of VMs increases, but once a low makespan is achieved, it doesn't decrease notably. In contrast, by incrementing the number of resources, the makespan slightly decreases but monetary cost increases proportional to the number of VMs.

With the exception of Epigenomics, evaluation function values for every case present similar behavior, they rise as the number of VMs increments, then it reaches a peak and finally drops. Peak time is presented when number of VMs is optimal for a particular case. For instance, when PSO-DS distributes Cybershake's tasks into four VMs it obtains an optimal case with a makespan of 39396 seconds at the cost of 6.908 dollars, even though the minimal achievable makespan is 34465 at the cost of 32.97 dollars. As for the Epigenomics, its evaluation function starts rising as a number of VMs are included, it reaches its peak with five VMs then it starts dropping but it suddenly has a rise with twelve VMs. The reason for this behavior is that the maximum number of parallel tasks is 24, for this reason PSO-DS converges with a uniform distribution of these parallel tasks among 12 VMs. The reason PSO-DS does not produce an improved scenario for 24 VMs is because the scheduling model contemplates an hourly-changing model of every VM, causing the configuration with 24 VMs to present the minimal makespan but its cost rises tremendously.

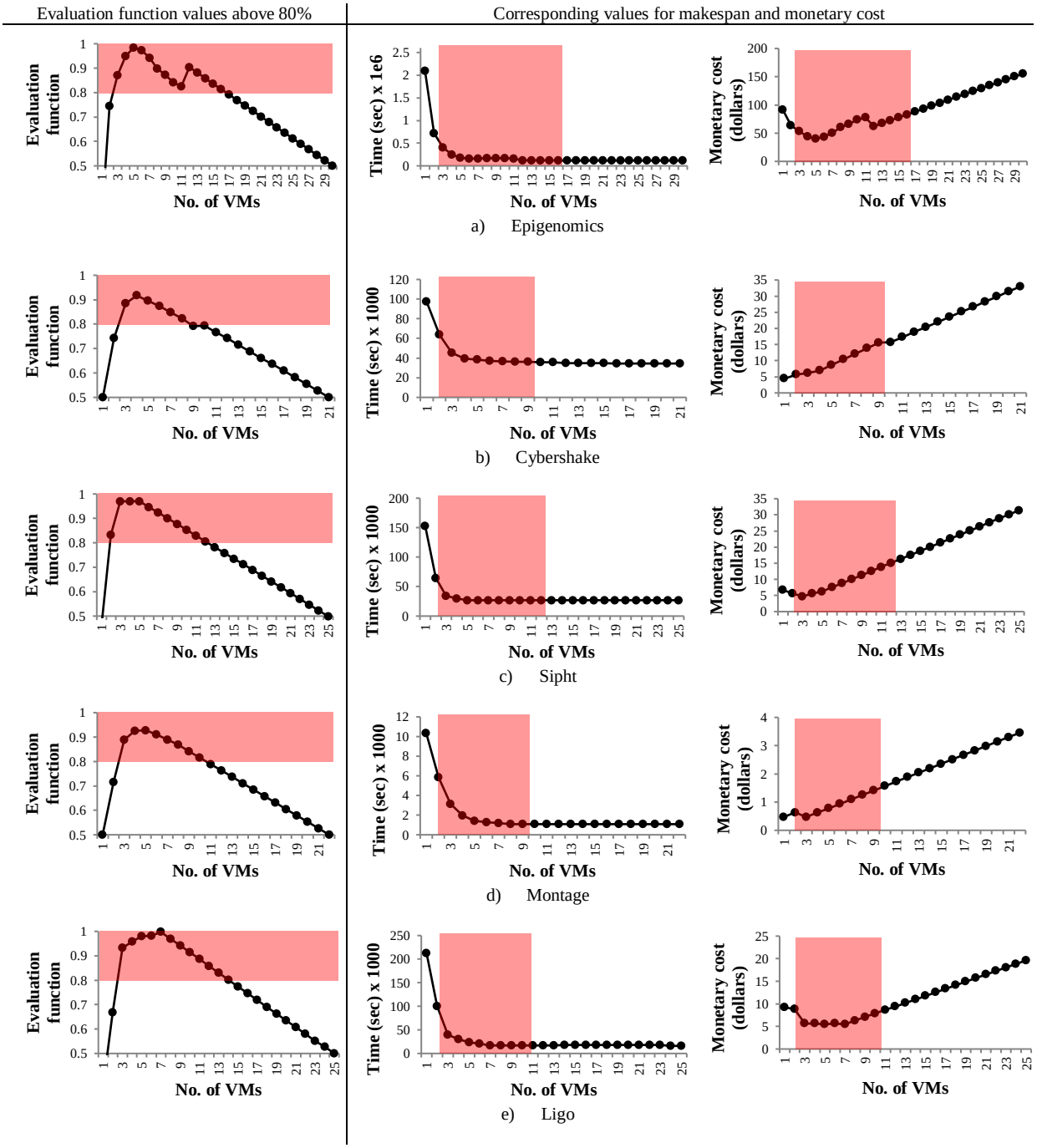


Figure 53. Results for function values, makespan and monetary cost for the five scientific workflows highlighting the area where function values are above 80%.

6.7 Analysis – Scheduling Large Workflows

In order to provide arguments for the competence of the PSO-DS, this experiment compares it against the four schedulers previously presented. With the exception of the Cybershake, results show the PSO-DS is able to provide better results especially for the cases

with a low number of VMs and function values above 80% as highlighted in the previous section.

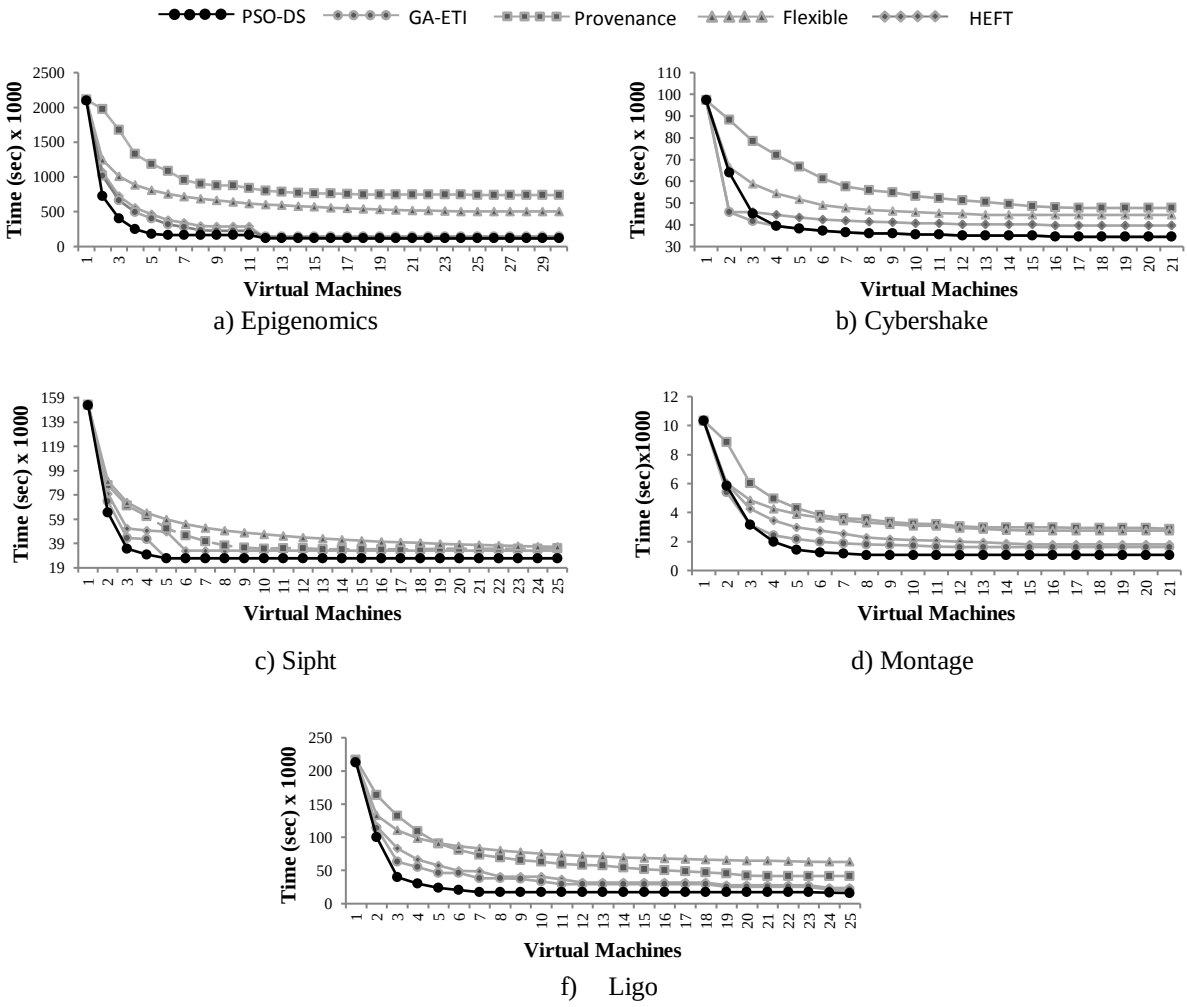


Figure 54. Execution time results with a different number of VMs.

An important reason for these positive results is that PSO-DS is designed to consider the most predominant factors affecting makespan such as task grouping that is based on their dependencies, file sizes and available number of VMs. It is important to highlight that whether HEFT and GA-ETI converge with similar makespan values they do it when the number of VMs increases. An important factor to emphasize is that optimal function values (above 80%) are presented as soon as makespan does the biggest drops. For example, in the Sipht result graph, in Figure 54c, the makespan drops from 64692 to 26879 seconds with two and five VMs respectively, beyond that number of resources makespan does not provide a

substantial improvement for any of the algorithms. This behavior is presented for the rest of the workflow execution.

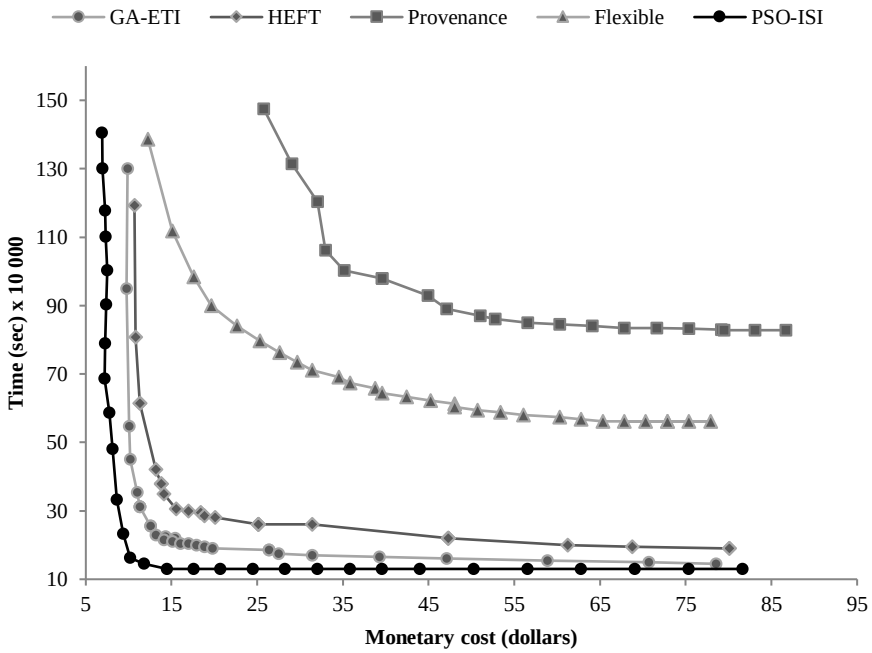


Figure 55. Epigenomics' $MSpn - MCst$ graph.

6.8 Discussion – PSO-DS Performance

This section provides a deeper analysis of the PSO-DS performance compared with the rest of the algorithms for solid reference. Figure 55 presents results of the five scheduling algorithms in a Pareto Front fashion, i.e. on the $MSpn$ vs $MCst$ graph. Pareto Frontier is a concept in economics with application in engineering [44]. It is defined as positioning individuals where no one of them can improve its position without deteriorating another's. The Pareto frontier is built from connecting such individuals forming a curved-graph which in engineering is used as a trade-off of values. In this case only the Epigenomics workflow is analyzed since the rest of the workflows present similar behavior. Figure 55 shows the results for makespan with its corresponding monetary cost. As readers will notice, the PSO-DS is able to converge to the values spread along the complete $MSpn$ vs $MCst$ curve. This analysis provides proof that PSO-DS does not get trapped in particular sections of the solution space and corroborates that the local best solution does not over-dominate the search for a solution. In a similar manner, GA-ETI and HEFT distribute their values with great proximity to the

PSO-DS due to their analysis and filtering of solutions. In contrast, FSV and Provenance missed the opportunity to provide superior results due to their minimal workflow analysis.

Table 11. Algorithms' parameters selection for comparing GA-ETI, HEFT, Provenance, Flexible, PSO-DS

Algorithm	**Parameter**			**Description of how each value allows an even comparison among scheduling algorithms**
	Name	Value	Description	
Flexible	Step	1	Number of increments to VM number	It allows the algorithm to include VMs one by one as the proposed algorithms
	M	234620	Time to execute with onle one VM	This is the time to execute application using a single VM
	MR	0	Marginal revenue	It starts from 0 to allow algorithm to analyze all possible values. estimation is done through the user's budget function and the average response time T , both of which are provided as input.
	MC	0.15	Marginal cost	We choose to have MC as an input parameter as we expect either the cloud administration or the IaaS consumer to set the cost function
	B	10.362	Budget (Monetary cost if executed with a single VM)	We allow the budget to be the one with a single VM. This is the monetary cost to execute application using a single VM
	physical nodes	30	Number of available VM	Previous experiments exhibited that using more than 30 machines does not improve execution objectives
	Users	1	Number of users	This thesis considers individual analysis
	Initial VMs	1	Initial number of resources	Experiment allows algorithm of using from 1 to 30 VMs
	Max. VMs	30	Maximum number of vms per user	
	VM cost	0.157	Cost per VM per hour	Cost applies for the 5 algorithms
Provenance	α_1	0.5	Execution time weight criterion	α_3 is assigned a value of 0 so α_1 and α_2 match w_1 and w_2 from GA-ETI and PSO-DS
	α_2	0.5	Monetary cost weight criterion	
	α_3	0	Reliability weight criterion	
	Input files	997	Input files	The assigned value matches with the number of tasks in application.
	Available virtual cores	30	Group of cores that the algorithms has access to	Similarly to the rest of the algorithms, 30 machines is the limit of available resources.
HEFT	SET_V	997	Number of task in the graph	This number matches the Epigenomics application
	SET_{CCR}	N/A	Communication to computation ratio	It doesn't apply, experiments uses task as defined in the Epigenomics application.
	SET_α	N/A	Graph shape parameter	Experiment uses the following task interdependencies: pipeline, data distribution, data aggregation and data redistribution
	SETβ	N/A	Range percentage of computation cost on processor	This parameter refers to the heterogeneity of vitual machiens. In this experiments each task has already set up an estimated time to be executed in a given machine
	Number of processors	30	Group of cores that the algorithms has access to	Similarly to the rest of the algorithms, 30 machines is the limit of available resources.
PSO-DS	P	100	Population	Preliminary tests exhibited that 100 particles give the best results.
	w_1	0.5	Makespan optimization weight	w_1 and w_2 have a value of 0.5 to give the same priority to monetary cost and execution time.

	w_2	0.5	Monetary cost optimization weight	
	c_1	2	Social acceleration coefficient	
	c_2	2	Personal acceleration coefficient	Preliminary experiments testing ω in the range of [0.9, 1.2] and $c_1 = c_2 = 2$ exhibited that a value of 1.2 for ω produces the best results.
	ω	1.2	Inertia coefficient	
	V_{max}	4	Velocity limit	V_{max} is set to 4 to prevent $s(v_{id}^t)$ in Eq. 30 to continuously approaching the upper and lower bound of [0,1].
GA-ETI	w_1	0.5	Makespan optimization weight	w_1 and w_2 have a value of 0.5 to give the same priority to monetary cost and execution time.
	w_2	0.5	Monetary cost optimization weight	
	p_c	0.6	Crossover probability	
	p_s	0.2	Swap mutation	
	p_{inc}	0.2	Increment mutation	Preliminary experiments exhited superior results with this value
	p_{dec}	0.2	Decrement mutation	
	IP	10	Pre-initial population factor	
	P	100	Initial population	It matches the PSO-DS initial population size

Table 11 presents selected parameters of each scheduling algorithm for making a match for comparison purposes. Every scheduling algorithm uses their internal process for producing a final solution. Then, graph in Figure 55 presents execution time and monetary cost values calculated with equations 21 and 22 for all algorithms to allow a uniform comparison.

Measure scheduling time

An important factor in scheduling algorithms is the time to run the scheduling process itself. Table 12 presents timeframes to produce a scheduling configuration for a workflow with 1000 nodes. The PSO-DS and GA-ETI present the highest processing time with 9500 and 50090 ms respectively while the FSV executes the algorithm in 15.1 ms. The reason for this large difference is that FSV does not base its scheduling approach on an evolution of solutions, it rather chooses a final solution from a limited number of configurations. Similarly, HEFT considers a single solution executing the algorithm in only 115.3 ms. In contrast, PSO-DS and GA-ETI evaluate a group of solutions on a series of iterations unfolding solutions and executing their algorithms on a larger timeframe. However, none of

the approaches has an excessive execution time compared with the final makespan as an exhibit on column three of Table 12.

Table 12. Scheduling time and its relation with final makespan

Scheduler	Scheduling time (ms)	Percentage of scheduling time compared with final makespan
PSO-DS	9500	9500 ms / 180889 s = ~0.00%
GA-ETI	50090	50090 ms / 250143 s = ~0.00%
HEFT	115.3	115.3 ms / 466189 s = ~0.00%
FSV	15.1	15.1 ms / 809208 s = ~0.00%
Provenance	154	154 ms / 1182699 s = ~0.00%

6.9 Summary

This chapter proposed PSO-DS, architecture to execute scientific workflows in cloud computing systems. PSO-DS is based on PSO with a special adaptation to the scheduling problem including a discrete formatting of particles and an enhanced super element. Using five workflows representing current scientific problems, PSO-DS was tested and proved its dominance against four cloud schedulers (HEFT, Provenance, FSV and GA-ETI). Through experimentation, PSO-DS highlighted the need for a specialized scheduler on top of WMS. PSO-DS is able to provide superior results in terms of makespan and monetary cost compared with other schedulers, in particular in the cases with a small number of resources providing function values above 80%. Additionally, PSO-DS provides scheduling configuration with values spread along the complete $MSpn$ vs $MCst$ curve. PSO-DS's positive results exhibited the main factors to consider during the scheduling process in order to optimize time and cost; such characteristics include task grouping, job dependencies, file sizes and available number of VMs. Additionally, PSO-DS exhibited that superior solutions execute parallel tasks sequentially on the same VM in order to lower file transferring. PSO-DS experiments underline the importance of not relaxing the monetary budget. Users may have an unlimited budget, or some schedulers may consider this assumption. By loosening the monetary budget, the user may obtain similar results at the expense of a pointless charge.

7 Discussion

This chapter presents a discussion of the obtained results in regards to the thesis objectives. Firstly, Section 7.1 gives an overview of the targeted thesis objectives. Then, Section 7.2 provides an analysis of findings and patterns. Section 7.3 presents a discussion of results. Subsequently, Section 7.4 discusses limitations of this research. Then, Section 7.5 provides recommendation for future investigation and finally, Section 7.6 presents a conclusion.

7.1 Preliminaries

This thesis firstly presented BaRRS, a scheduling approach that minimizes execution time and monetary cost; secondly it explored GA-ETI, a scheduling approach based on the genetic algorithms that converge to a final solution after considering a complete solution space; finally it presented PSO-DS, a scheduling mechanism with a reduced complexity algorithm that includes all the features from previous scheduling approaches.

BaRRS: It gives a solution to the scheduling problem by minimizing application execution time and monetary cost. It is a mechanism based on load balancing, data replication and additionally it explodes parallelism to maximize system utilization. Moreover, BaRRS is capable of building a trade-off considering scenarios with different values of execution time and monetary cost.

GA-ETI: It is a scheduling mechanism capable of adapting the number of resources to a particular workflow execution. This approach is based on the capabilities of the genetic evolution theory in providing findings in respect to runtime and monetary cost. Moreover, GA-ETI decreases randomness from the original genetic process that enables the GA-ETI developed here to converge on the final solution without running the algorithm over excessive iterations.

PSO-DS: The PSO-DS is capable of manipulating workflow with a high number of tasks and data files. PSO-DS is an adaptation of the PSO mechanism to construct a scheduler that can converge to a final solution in a record time. Different to other PSO adaptations, the PSO-DS produces an enhanced particle reconstruction and presents a Super Particle that allows the algorithm to reduce the search time.

Tables Table 13 and Table 14 presents the execution time and monetary cost results of BaRRS, GA-ETI, PSO-DS, HEFT, Provenance, Flexible and Pegasus-WMS for the scheduling of the five analysed workflows. The proposed algorithms predominate the lowest results among the rest of the algorithms due to their techniques for considering specific application characteristics such as large data files and task dependencies.

Table 13. Execution time comparison between scheduling algorithms.

	HEFT (sec)	**Prove-nance** (sec)	**Flexible** (sec)	**GA-ETI** (sec)	**PSO-DS** (sec)	**BaRRS** (sec)	**PEGASUS WMS** (sec)
Epigenomics	42163	26450	98320	29846	6316	30843	78759
Montage	252	393	475	279	230	360	3464
Cybershake	4713	5805	6049	3056	2769	3389	4903
SHIPT	3687	4176	6532	4428	2587	4181	6428
Ligo	2846	3372	6944	4023	2346	2676	8157

Table 14. Monetary cost comparison between scheduling algorithms.

	HEFT (dlls)	**Prove-nance** (dlls)	**Flexible** (dlls)	**GA-ETI** (dlls)	**PSO-DS** (dlls)	**BaRRS** (dlls)	**PEGASUS WMS** (dlls)
Epigenomics	13.07	35.63	16.38	9.49	5.75	14.13	34.92
Montage	0.92	1.09	0.52	0.20	0.25	0.20	1.10
Cybershake	2.07	1.61	1.54	0.67	0.53	0.38	0.38
SHIPT	1.70	2.47	1.44	0.53	0.84	0.53	1.19
Ligo	1.24	2.35	2.61	1.54	0.92	1.70	3.50

7.2 Analysis of findings and patterns identified in experimental results

Workflow tasks parallelism

Experimentation demonstrates that intrinsic characteristics of applications play an important role in the scheduling process. For instance, the Epigenomics application that exhibits parallelism on 96% of its nodes has the potential to employ 24 VMs for its execution. Nevertheless, when HEFT and GA-ETI distribute these tasks among 12 VMs, the execution time reaches almost the minimum runtime as shown in Figure 43a. Notice that doubling the numbers of resources to 24 does not necessarily reduce execution time by half. This is because parallel tasks require transfer of replica files causing extra file transfer that increments runtime.

The aforementioned findings imply that GA-ETI and other scheduling algorithms such as [7, 13, 72] do not necessarily exploit parallelism to the fullest. For instance refer to the results of the Cybershake workflow from
Figure 43b. This is a data orientated workflow with its majority of nodes distributed on two workflow levels. These nodes can be distributed over 48 resources but instead the GA-ETI selects only two VMs. The reason for this decision is that the time to transfer files is significant greater than the execution of tasks in the workflow.

Nevertheless the GA-ETI exhibited a particular favouritism for specific solutions. Figure 44 expresses results from Figure 43a in a pareto style. On it, the GA-ETI shows that the majority of solutions are concentrated on the center of the graph. An important implication of these findings is that configurations have 6, 8, and 12 machines that completely exploit application parallelism.

System utilization

The results obtained in Sections 4.6 4.7 showed that system utilization has a direct relation to task parallelism. The data replication technique from BaRRS produces a strong influence on parallel tasks, consequently the system utilization increases. To illustrate this refer to Figure 19 Figure 22 where BaRRS and Provenance [72] produce trade-off for applications exhibiting parallelism. On them, the proposed BaRRS presented superior outcomes. The reason that Provenance does not produce similar results is that it allows users to freely manipulate monetary constraints without advising of the potential degradation in the

system utilization. On the contrary, the proposed BaRRS allows a user to change the priority to the monetary cost optimization but it also keeps control of the runtime; as a consequence, the system utilization is not affected.

An additional finding regarding the system utilization is the strong dependence of this parameter on the nature of each application. The tested workflows have the same number of tasks but each one requires different computational power and consequently their analysis is different. To illustrate this refer to Figure 23,Figure 26,Figure 29 andFigure 32, graphs exhibit different shapes for each workflow type. The shape of these graphs is dictated by the number of data files, tasks dependencies and computational demands of each workflow.

Number of resources

This thesis makes a contribution to open discussion about the number of resources to execute applications. Results from Section 4.7 exhibited that the number of resources directly affects the scheduling process. The number of nodes in an application does not directly influence the final number of resources as exhibited in Table 7. For this reason, the number of resources was included as a variable in the GA-ETI. This approach provides an interesting tactic to solve the problem of selecting the number of resources to run a workflow based on its nature.

Randomness of the genetic algorithm

An important contribution of GA-ETI was the reduction of randomness compared to the original GA as presented in Figure 45. This contribution relies on the GA-ETI mutation operator. This operator introduces additional resources into the internal genetic process causing diversity among chromosomes. This diversity causes the GA-ETI to expand the range of scheduling options. A deep analysis of chromosome evolution in the GA-ETI revealed that after each generation the quality of each chromosome significantly increments converging to a final solution with fewer iterations.

A trade-off of optimization objectives

This thesis introduced a novel view to build a trade-off that a user employs to decide which of the optimization objectives must have higher priority. The objective of the BaRRS

trade-off is to offer the user a general view to execute an application considering its characteristics such as computational demands and file data usage. Each of the different options from the trade-off offers advantages and disadvantages that only a user can decide based on his/her particular interests. To illustrate this concept Figure 19Figure 22 present a trade-off exhibiting the different options to execute each application. Figure 19 shows that a user has the option to execute application at a minimal time of ~20 000 seconds at the cost of 20.74 dollars or reduce monetary cost to 10.676 dollars at the expense of increasing execution time to ~120 000 seconds. The main objective of the BaRRS is to guide a user in selecting the best option according to his/her needs.

System utilization and evaluation function values

Another interesting finding was the similarities between system utilization and evaluation function. Experimentation in Section 6.7 shows that not all results exhibit an evaluation function value above 80% when testing the PSO-DS as shown in Figure 53. Shapes of these graphs have strong similarities with the system utilization results from Section 4.7 . Readers must also notice that the number of machines in the shaded area in columns 2 and 3 from Figure 53 matches the ones that obtained the highest evaluation function values in column 1 from the same figure. Moreover, runtime and monetary cost uniformly decrease and increase, respectively, as the number of resources increases. Nevertheless, only the system utilization exhibits a peak at a particular number of machines (see Figure 23Figure 26 and Figure 32).

7.3 Important findings

One of the important findings of this thesis is the time to schedule large applications with the PSO-DS. The overhead caused by this scheduling process represented an insignificant percentage of the total runtime for applications with a greater number of tasks. This result positions the PSO-DS as a solution to the problem to schedule large applications.

As presented in Table 10 (Section 6.8 , the PSO-DS algorithm produces a scheduling solution in 9.500 seconds for an application with a thousand nodes that executes in 180889 seconds while the FSV, a scheduling algorithm based on [7], executes the algorithm in 15.1 ms. The reason for this large difference is that the FSV algorithm is not an evolution/iteration

of solutions, instead it builds a final solution from a partial section of the solution space. Correspondingly, HEFT [13] produces a single solution out of a pre-ordered list.

In contrast, PSO-DS and GA-ETI assess a collection of solutions over a number of iterations. For this reason, PSO-DS and GA-ETI execute their algorithms on a larger timeframe. Nevertheless, none of the approaches has an excessive execution time compared with the final makespan. Furthermore, the high scheduling overhead time as in PSO-DS yields a reduced final makespan. This is because PSO-DS takes advantage of the PSO capabilities; in addition it uses a reconstruction particle that allows algorithms to build solutions that are impossible to build with traditional adaptations of the PSO to the scheduling problem.

7.4 Limitations

The proposed algorithms require different inputs such as the VM characteristics including number of cores, bandwidth and memory size. This limitation may prevent algorithms from an accurate analysis. Nevertheless, most of the cloud computing providers make available this information. Moreover, a history module can be added to the scheduler so it stores information from a cloud computing provider. These values are compulsory for producing an accurate estimation of application execution. From one side, the bandwidth value is employed for estimating file transfer times. File transfers become critical when analysing data intensive applications. From the other side, the number of cores and memory size affect directly task computation time and the way scheduler distribute tasks on the VMs.

The scheduling algorithms developed in Chapters 4 5 6 are susceptible to execution interruptions. The three proposed scheduling algorithms, BaRRS, GA-ETI and PSO-DS are designed to produce a static scheduling algorithm. Whenever a user halts a workflow execution it affects the scheduling plan. After resuming execution, the final makespan may have a significant difference to the one projected by the scheduler. The main reason for these possible differences is that data files are programmed to be transferred, saved or deleted; by halting execution, the workflow manager needs to rearrange such data files affecting the produced scheduling plan. To avoid such inconveniences, the scheduler must produce a new plan for the remaining tasks.

7.5 Recommendation for Further Research

Through the extensive experimentation to produce this thesis, the author noticed that the workflow manager system does not always detect when a machine halt task execution. These interruptions directly affect the scheduling plan. For this reason it is recommended to include an execution watcher capable of detecting machines that do not follow the scheduling plan, for those scenarios in which the scheduler must re-schedule affected tasks.

7.6 Summary

This thesis presented efficient scheduling algorithms that directly benefit the client and provider from a cloud computing system environment. From one side, the client is able to execute his application at a lower time and cost compared with other scheduling algorithms and more importantly, the client is able to prioritize objectives according to his/her needs. From the other side, the cloud provider is able to efficiently administrate its computing resources and is able to maintain a high level of satisfaction to its clients. Similarly, the environment benefits from reduced efficient energy consumption.

8 Conclusions

This chapter presents the conclusion of this thesis. It summarizes the main contributions and results in Section 8.1 . Firstly, Subsection 8.1.1 presents the main highlights from the BaRRS contribution. Subsection 8.1.2 summarizes the contribution from GA-ETI and provides conclusions for selected experiment results. Subsequently, Section 8.1.3 summarizes the PSO-DS algorithm. Finally, Section 8.2 discusses the direction of future work.

8.1 Contributions and Summary of Results

This thesis focused its efforts to incrementing our knowledge of the workflow scheduling problem. One of the main focuses of this study was to examine key factors to increase scheduling efficiency, such factors have not been properly analyzed by current scheduling frameworks despite the great attention the research community has put to this problem. Approaches presented in this thesis contribute to identifying the limitations of current scheduling frameworks and to produce innovative techniques to engage these issues.

8.1.1 Balanced and File Reuse-Replication Scheduling

Chapter 4 presented BaRRS, a scheduling engine combining three distribution mechanisms to give solutions to the scheduling problem. BaRRS encompasses load

balancing, data reuse and replication techniques to build an efficient heuristic to distribute a workflow's tasks to cloud computer resources. Data replication and data reuse complement each other giving BaRRS the opportunity to discover opportunities in parallelism and reduce file transfers. The load balancing mechanism contributes to increment system utilization. To select the scheme that fulfills user needs, BaRRS provided a tradeoff with the execution time and monetary cost.

Data replication, data reuse and load balancing techniques

Firstly, file replication is a consequence of executing parallel tasks. It consists in creating and transferring a replica of a file. Even though other proposals have already included data replication, the present study examines the effect of over exploiting parallelization. This study analyzes the relation between file transfers, total execution time and monetary cost. Secondly, to fully obtain the benefits of data replication, BaRRS complements it with a data reuse technique. The file reutilization mechanism reduces the number of file transfers during workflow execution. This technique assigns parent and descended tasks into the same VM, consequently the file transfer is not needed. Whenever the time to transfer the input files of a task exceeds its execution time, it applies data reutilization, i.e. it adds tasks to the same resource executing its task parent, otherwise the task is added to another VM, causing a new transfer (replication). Finally, data replication and reuse techniques can overload a particular resource. The load balancing objective is to evenly distribute tasks among all VM queues. This procedure interchanges tasks between all queues without worsening any optimization goal.

Tradeoff based on a Pareto Frontier style

BaRRS produces a tradeoff with the values of monetary costs and execution time to execute scientific workflows. In order to decrease the scheduling overhead time, this algorithm produces a fixed number of estimations then it uses the exponential function to build the complete tradeoff graph. BaRRS indirectly changes the number of VMs in order to present different time and cost configurations. The tradeoff analysis is modeled by the exponential shape graph to create a set of exhaustive scheduling configurations. A trade-off graph is defined as set of solutions where each solution has a different number of VMs with

its respective execution time and monetary cost. This trade-off follows a similar shape as a Pareto frontier, the main difference is that trade-off offers flexibility to analyze the complete spectrum of number of VMs.

Summary of results

Results show that BaRRS achieves lower execution time compared to Provenance. Experiments showed that solutions with a higher number of VMs do not necessarily lead to lower execution times nor monetary cost. Certainly, solutions with a high number of VMs presented a high monetary cost without necessarily reducing runtime. A close examination revealed that achieving a minimum execution time is less demanding than obtaining a minimum monetary cost. This is a consequence of the efficient application of the triple scheduling mechanism to reduce execution time.

8.1.2 Genetic Algorithm with Efficient Tune-In of Resources

The VM selection problem

In Chapter 5 this study expressed the problem to select the optimal number of resources to execute workflows by incorporating BaRRS techniques into a modified evolutionary algorithm. GA-ETI, the proposed approach, gives a solution to this problem by adapting the GA. The GA is a robust technique to solve complex problems in engineering and science due to its capabilities to recognize a global optimum in the complete search space.

The VM selection challenge

Although other solutions have applied the capabilities of the GA, the solution proposed here: (1) carefully adapted and modified the crossover and mutation operators to reduce randomness in the GA, (2) its modified mutation operator directly changes the number of resources in chromosomes and (3) it converges to a final solution with a reduced number of generations. In order to control the number of VMs, each chromosome in the GA-ETI model encompasses a scheduling configuration. Genes encode tasks and their respective VM for execution. In this way, the mutation operator is able to identify a VM for its deletion or

add a new one to a given chromosome. Results demonstrated that the number of tasks in a workflow does not influence the final number of resources that an application needs. Instead the number of VMs has a stronger relation to (1) workflow task dependencies, (2) file transfer demands, and (3) task dependency constraints.

The GA-ETI results highlighted the need to analyze task dependencies. For instance, the GA-ETI converged to solutions where groups of parallel tasks are allocated to a single VM. These decisions lead the GA-ETI to converge to solutions with a lower number of VMs compared with the number of tasks that can run in parallel. These decisions also contributed to reduce the number of file transfers as the VM required only one set of files per group of parallel tasks. To clarify this concept, in the Cybershake application, the GA-ETI executes groups of four parallel tasks on the same VM reducing to ¼ of time the transfer time for the required tasks. In contrast, HEFT does not consider this option due to its scheduling policies focusing on a task at the time. Results also demonstrated that incrementing the number of resources to maximum does not necessarily increase efficiency. For example, the Cybershake workflow has two groups of 48 parallel tasks and the GA-ETI converges to solutions that only require eight VMs. To clarify this matter, the study went into a deep observation of the GA-ETI process, and observed that chromosomes that exploit parallelization with 12, 24 and 48 VMs did not present a significant execution time reduction but their monetary cost was significantly greater. The time execution could not be reduced due to the high number of file transfers. Finally, GA-ETI introduces a new policy to predefine the number of VMs for a given workflow. It first identifies the maximum number of tasks that can run in parallel. Then it limits the number of VMs to that specific number since an extra VM would remain idle.

The impact of the GA on the scheduling problem

Besides the benefit in selecting the number of VMs, the modified operators of the GA-ETI allow the algorithm to converge to a final solution with fewer generations compared with the original GA. Firstly, the clustered crossover prevents the algorithm from breaking chromosomes at a random point, instead it guides the algorithm to not destroy specific areas of the chromosomes that have an optimal distribution. Secondly, the increment/decrement mutation adds new VM to the population that allows the GA-ETI to restructure a particular chromosome. Overall, both mechanisms complement each other, adapting the original GA

into a powerful tool with a reduced use of randomness causing the algorithm to converge to a final solution at a reduced number of iterations.

Results also exhibited that the algorithm reaches a minimum execution time within a few generations; and that the execution time of the final solution does not present a big difference with this value. In contrast, the number of VMs reduces tremendously between the start and end of the algorithm. This is because at the start of the algorithm most of chromosomes have as many VMs as parallel tasks. Then, as the algorithm continues, the main challenge is to reduce/maintain a low execution time while reducing the number of VMs to obtain a satisfactory monetary cost.

Summary of results

This study concludes that parallelization, an important feature of distributed systems, is exploited inefficiently in many schedulers. From one side, the number of VMs has a strong influence on the scheduling configuration to execute applications. On the other side, the number of VMs is driven by workflow characteristics. Current schedulers do not consider the second premise. As a result their scheduling configurations are based on unsolid data. A few exceptions consider this problem, offering simplistic solutions without a deep analysis. All other approaches use a fixed pool of resources and no discussion exists to guide users in selecting this parameter.

8.1.3 Particle Swarm Optimization with Discrete Adaptation and a Featured SuperBEST

In Chapter 6 this study explored the problem of analyzing larger workflows without (exponentially) incrementing the scheduling overhead time. The PSO-DS, the solution proposed here, is an adaptation of the original discrete PSO to solve the scheduling problem. The PSO is an optimizer with similar capabilities as the genetic algorithm to produce optimal results. The most important characteristic of the PSO is its low complexity process. More specifically, a scheduling algorithm based on the PSO converges to finest solutions after considering a great number of combinations without requiring excessive processing time.

Although other studies have applied the PSO to the scheduling problem, this study's proposed solution: introduced a new particle named SuperBest to build the global best

element, it also introduced a novel scheduling reconstruction from particles velocities that permits to assign sets of consecutive tasks to the same VM. The evaluation of the PSO-DS aimed to compare it with four other mechanisms that have equivalent objectives to provide arguments to validate its performance. The performance metrics were: scheduling time, makespan and monetary cost. In greater detail, this study analyzed the mentioned metrics focusing on the areas where the system utilization is greater than 80%. Results validate the efficiency of the algorithm when analyzing large workflows as well as the evidence algorithm maintain a low/reasonable scheduling overhead time.

The Algorithm's main contributions

Firstly, experimentation showed that PSO-DS achieved improved results compared with the HEFT, Provenance and FSV. These outcomes are the result of a series of reasons, first of all the PSO-DS converges to solutions where machines are used completely for the time they are hired. This is a consequence of considering monetary cost as an optimization objective. Additionally, a deeper observation of experiment results revealed that PSO-DS uniformly distributes tasks among resources for the cases where tasks have a long execution time.

PSO-DS presented a higher scheduling overhead time compared with the HEFT, even though its scheduling solution yields a reduced final makespan. For instance for a workflow with a thousand nodes, PSO-DS needed 9500 ms to run the algorithm while HEFT runs in only 115.3 ms. Even so, PSO-DS achieves a final makespan of 180889 seconds while HEFT achieves 466189 seconds. The reason for this gap is that PSO-DS evaluates a population of solutions on a series of iterations while HEFT only analyzes a single solution configuration.

The impact of the PSO to the scheduling problem

In addition, experimentation showed that PSO-DS results are spread along the $MSpn$ vs $MCst$ curve in a Pareto style. With this analysis, the present study provided proof that PSO-DS does not get trapped in particular sections of the solution space. Additionally, it corroborates that local best solution does not over dominate the search for a solution. Similarly, the design of the PSO-DS considers the most predominant factors that affect makespan such as task grouping, file sizes and available number of VMs.

Furthermore, the introduction of the SuperBest in the process to build the global best element provided warranty that the global best particle is predominant on the population. As a result, PSO-DS is able to provide better results even for cases with a low number of VMs. As for the scheduling reconstruction of particles velocities, the algorithm presented here permits sets of consecutive tasks to be assigned to the same VM in order to avoid unnecessary file transfers. For instance, in applications with moderate parallelization, PSO-DS demonstrated that superior solutions execute parallel tasks sequentially on the same VM in order to reduce file transferring.

Summary of results

The PSO-DS underlines the importance of not relaxing monetary budget. Users may have an unlimited budget, or some schedulers may consider this assumption. But by loosening tge monetary budget, a user may obtain similar results at the expense of a pointless charge. Results showed that the algorithm is capable of converging on a final solution at a lower number of iterations. More precisely, PSO-DS converges to solutions where groups of parallel tasks are executed on the same VM, consequently PSO-DS is able to decrease the number of data transfers.

8.2 Future Work

Cloud computing providers offer a wide number of services. Furthermore, more complicated systems such as federated clouds and multi-cloud environments significantly increment the number of such services making more complex the selection of services and administration of computing resources. Future work proposes a framework for addressing this problematic. The aim of this proposed framework is guiding users in selecting an adequate service for his/her needs, estimate performance and guide him/her through all the process of executing an application. The purpose of the framework is to act as a manager of applications offering a compatible solution for the different cloud services available. The objective is to create a tool that act as an interface between users and cloud computing resources. This project requires adapting our proposed scheduling algorithms into the framework and develops the required tools for creating an application manager. The main goal is using this

tool as an application manager in public cloud environments and subsequently for more complicated systems such as federated clouds and multi cloud environments. In a multi-clouds system, the framework can preserve control over files transfers across time zones and geographies areas. In a federated cloud, this framework will automate decisions of replicating files for exploiting parallelization.

Additionally, the experiments analyzed in this thesis reveal two important threads for future investigation. Firstly, since workflows' dependency patterns have an important influence on selection of scheduling policies, it is crucial to modify scheduling algorithms to accept a diverse set of patterns. The researcher's future direction is to address this issue and provide enhanced scheduling tools. Secondly, scheduling overhead time can impact feasibility of scheduling strategies especially for large size workflows. For that reason, future investigation are required to filter (detect and dismiss) low quality solutions before the scheduler analyses the application; this should significantly reduce the scheduling overhead time. Additionally, the researcher aims to develop/incorporate cloud pricing models to consider fluctuation of the hiring cost of VMs during scheduling. Also, a focus on performance oscillation in cloud environments and its impact on the execution of applications should be considered. Finally, the author of this thesis is focusing on the analysis of large sets of files (Big Data) for applications associated with biological viruses, terrorism and economic crisis behavior.

9 References

[1] I. Casas, J. Taheri, R. Ranjan, L. Wang, and A. Y. Zomaya, "A balanced scheduler with data reuse and replication for scientific workflows in cloud computing systems," *Future Generation Computer Systems*, 2016.

[2] I. Casas, J. Taheri, R. Ranjan, L. Wang, and A. Y. Zomaya, "GA-ETI: An Enhanced Genetic Algorithm for the Scheduling of Scientific Workflows in Cloud Environments," *Journal of Computational Science*, 2016.

[3] I. Casas, J. Taheri, R. Ranjan, and A. Y. Zomaya, "PSO-DS: A Scheduling Engine for Scientific Workflows Managers," *Journal of Supercomputing (Under revision)*, 2016.

[4] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Characterization of scientific workflows," in *Proceedings of the 3rd Workshop on Workflows in Support of Large-Scale Science (WORKS 2008)*, 2008, pp. 1-10.

[5] H. Kloh, B. Schulze, R. Pinto, and A. Mury, "A bi‐criteria scheduling process with CoS support on grids and clouds," *Concurrency and Computation: Practice and Experience*, vol. 24, pp. 1443-1460, 2012.

[6] I. A. Moschakis and H. D. Karatza, "Evaluation of gang scheduling performance and cost in a cloud computing system," *The Journal of Supercomputing*, vol. 59, pp. 975-992, 2012.

[7] K. Tsakalozos, H. Kllapi, E. Sitaridi, M. Roussopoulos, D. Paparas, and A. Delis, "Flexible use of cloud resources through profit maximization and price discrimination," in *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, 2011, pp. 75-86.

[8] C. Anglano and M. Canonico, "Scheduling algorithms for multiple bag-of-task applications on desktop grids: A knowledge-free approach," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, 2008, pp. 1-8.

[9] A. Sulistio and R. Buyya, "A time optimization algorithm for scheduling bag-of-task applications in auction-based proportional share systems," in *Computer Architecture and High Performance Computing, 2005. SBAC-PAD 2005. 17th International Symposium on*, 2005, pp. 235-242.

[10] A. Radulescu, C. Nicolescu, and P. P. Jonker, "CPR: Mixed task and data parallel scheduling for distributed systems," in *Parallel and Distributed Processing Symposium., Proceedings 15th International*, 2001, p. 9 pp.

[11] T. N'Takpe and F. Suter, "Critical path and area based scheduling of parallel task graphs on heterogeneous platforms," in *12th International Conference on Parallel and Distributed Systems-(ICPADS'06)*, 2006, p. 8 pp.

[12] Y.-K. Kwok, "Parallel program execution on a heterogeneous PC cluster using task duplication," in *Heterogeneous Computing Workshop, 2000.(HCW 2000) Proceedings. 9th*, 2000, pp. 364-374.

[13] H. Topcuoglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 13, pp. 260-274, 2002.

[14] Y. C. Lee and A. Y. Zomaya, "Stretch Out and Compact: Workflow Scheduling with Resource Abundance," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, 2013, pp. 219-226.

[15] *AWS | Amazon Elastic Compute Cloud (EC2)*. Available: http://aws.amazon.com/ec2/

[16] A. Bala and I. Chana, "A survey of various workflow scheduling algorithms in cloud environment," in *2nd National Conference on Information and Communication Technology (NCICT)*, 2011, pp. 26-30.

[17] M. Naghibzadeh, "Modeling Workflow of Tasks and Task Interaction Graphs to Schedule on the Cloud," *CLOUD COMPUTING 2016*, p. 81, 2016.

[18] A. Balmin, K. W. Hildrum, V. Nagarajan, and J. L. Wolf, "Automated scheduling management of MapReduce flow-graph applications," ed: Google Patents, 2016.

[19] N. Mishra, A. Singh, S. Kumari, K. Govindan, and S. I. Ali, "Cloud-based multi-agent architecture for effective planning and scheduling of distributed manufacturing," *International Journal of Production Research*, vol. 54, pp. 7115-7128, 2016.

[20] H. M. Al-Najjar and S. S. N. A. S. Hassan, "A survey of job scheduling algorithms in distributed environment," in *Control System, Computing and Engineering (ICCSCE), 2016 6th IEEE International Conference on*, 2016, pp. 39-44.

[21] K. Wang, K. Qiao, I. Sadooghi, X. Zhou, T. Li, M. Lang, and I. Raicu, "Load‐balanced and locality‐aware scheduling for data‐intensive workloads at extreme scales," *Concurrency and Computation: Practice and Experience*, vol. 28, pp. 70-94, 2016.

[22] H. Qu, O. Mashayekhi, D. Terei, and P. Levis, "Canary: A Scheduling Architecture for High Performance Cloud Computing," *arXiv preprint arXiv:1602.01412*, 2016.

[23] J. Y. Leung, *Handbook of scheduling: algorithms, models, and performance analysis*: CRC Press, 2004.

[24] P. Lubomski, A. Kalinowski, and H. Krawczyk, "Multi-level Virtualization and Its Impact on System Performance in Cloud Computing," in *International Conference on Computer Networks*, 2016, pp. 247-259.

[25] J. W. Rittinghouse and J. F. Ransome, *Cloud computing: implementation, management, and security*: CRC press, 2016.

[26] S. Goswami and A. Das, "Optimization of Workload Scheduling in Computational Grid," in *Proceedings of the 5th International Conference on Frontiers in Intelligent Computing: Theory and Applications*, 2017, pp. 417-424.

[27] G. L. Stavrinides and H. D. Karatza, "Scheduling Different Types of Applications in a SaaS Cloud," in *Proceedings of the 6th International Symposium on Business Modeling and Software Design (BMSD'16)*, 2016, pp. 144-151.

[28] HTCondor. *High Throughput Computing*. Available: http://research.cs.wisc.edu/htcondor/

[29] M. Maheswaran, S. Ali, H. Siegal, D. Hensgen, and R. F. Freund, "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems," in *Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth*, 1999, pp. 30-44.

[30] M. Rahman, R. Hassan, R. Ranjan, and R. Buyya, "Adaptive workflow scheduling for dynamic grid and cloud computing environment," *Concurrency and Computation: Practice and Experience*, vol. 25, pp. 1816-1842, 2013.

[31] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, and J. Good, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, pp. 219-237, 2005.

[32] R. Y. Rubinstein and D. P. Kroese, *Simulation and the Monte Carlo method*: John Wiley & Sons, 2016.

[33] P. Dagum, R. Karp, M. Luby, and S. Ross, "An optimal algorithm for Monte Carlo estimation," *SIAM Journal on computing*, vol. 29, pp. 1484-1496, 2000.

[34] M. HoseinyFarahabady, Y. C. Lee, and A. Y. Zomaya, "Randomized approximation scheme for resource allocation in hybrid-cloud environment," *The Journal of Supercomputing*, vol. 69, pp. 576-592, 2014.

[35] K. Boston and P. Bettinger, "An analysis of Monte Carlo integer programming, simulated annealing, and tabu search heuristics for solving spatial harvest scheduling problems," *Forest science*, vol. 45, pp. 292-301, 1999.

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

[37] C. Bierwirth and J. Kuhpfahl, "Extended GRASP for the Job Shop Scheduling Problem with Total Weighted Tardiness Objective," *European Journal of Operational Research*, 2017.

[38] W. F. Boyer and G. S. Hura, "Non-evolutionary algorithm for scheduling dependent tasks in distributed heterogeneous computing environments," *Journal of Parallel and Distributed Computing*, vol. 65, pp. 1035-1046, 2005.

[39] S. AlEbrahim and I. Ahmad, "Task scheduling for heterogeneous computing systems," *The Journal of Supercomputing*, pp. 1-26, 2016.

[40] T. Ma and R. Buyya, "Critical-path and priority based algorithms for scheduling workflows with parameter sweep tasks on global grids," in *17th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'05)*, 2005, pp. 251-258.

[41] T. Wang, Z. Liu, Y. Chen, Y. Xu, and X. Dai, "Load Balancing Task Scheduling Based on Genetic Algorithm in Cloud Computing," in *Dependable, Autonomic and Secure Computing (DASC), 2014 IEEE 12th International Conference on*, 2014, pp. 146-152.

[42] C.-L. Chen, V. S. Vempati, and N. Aljaber, "An application of genetic algorithms for flow shop problems," *European Journal of Operational Research*, vol. 80, pp. 389-396, 1995.

[43] A. R. Simpson, G. C. Dandy, and L. J. Murphy, "Genetic algorithms compared to other techniques for pipe optimization," *Journal of water resources planning and management*, vol. 120, pp. 423-443, 1994.

[44] J. Taheri, A. Y. Zomaya, and S. U. Khan, "Genetic algorithm in finding Pareto frontier of optimizing data transfer versus job execution in grids," *Concurrency and Computation: Practice and Experience*, 2012.

[45] K.-S. Tang, K.-F. Man, S. Kwong, and Q. He, "Genetic algorithms and their applications," *Signal Processing Magazine, IEEE*, vol. 13, pp. 22-37, 1996.

[46] Z. Zheng, R. Wang, H. Zhong, and X. Zhang, "An approach for cloud resource scheduling based on Parallel Genetic Algorithm," in *Computer Research and Development (ICCRD), 2011 3rd International Conference on*, 2011, pp. 444-447.

[47] K. Zhu, H. Song, L. Liu, J. Gao, and G. Cheng, "Hybrid genetic algorithm for cloud computing applications," in *Services Computing Conference (APSCC), 2011 IEEE Asia-Pacific*, 2011, pp. 182-187.

[48] L. Liu, M. Zhang, R. Buyya, and Q. Fan, "Deadline‐constrained coevolutionary genetic algorithm for scientific workflow scheduling in cloud computing," *Concurrency and Computation: Practice and Experience*, 2016.

[49] G. Luo, X. Wen, H. Li, W. Ming, and G. Xie, "An effective multi-objective genetic algorithm based on immune principle and external archive for multi-objective integrated process planning and scheduling," *The International Journal of Advanced Manufacturing Technology*, pp. 1-14, 2017.

[50] B. M. Varghese and R. J. S. Raj, "A survey on variants of genetic algorithm for scheduling workflow of tasks," in *Science Technology Engineering and Management (ICONSTEM), Second International Conference on*, 2016, pp. 489-492.

[51] R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proceedings of the sixth international symposium on micro machine and human science*, 1995, pp. 39-43.

[52] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in *Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on*, 1997, pp. 4104-4108.

[53] B. Jarboui, N. Damak, P. Siarry, and A. Rebai, "A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems," *Applied Mathematics and Computation*, vol. 195, pp. 299-308, 2008.

[54] J. Kennedy, "Particle swarm optimization," in *Encyclopedia of machine learning*, ed: Springer, 2011, pp. 760-766.

[55] C.-J. Liao, C.-T. Tseng, and P. Luarn, "A discrete version of particle swarm optimization for flowshop scheduling problems," *Computers & Operations Research*, vol. 34, pp. 3099-3111, 2007.

[56] S. Pandey, L. Wu, S. M. Guru, and 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*, 2010, pp. 400-407.

[57] D. Sha and C.-Y. Hsu, "A hybrid particle swarm optimization for job shop scheduling problem," *Computers & Industrial Engineering*, vol. 51, pp. 791-808, 2006.

[58] K. R. Kumari, P. Sengottuvelan, and J. Shanthini, "A Hybrid Approach of Genetic Algorithm and Multi Objective PSO Task Scheduling in Cloud Computing," *Asian Journal of Research in Social Sciences and Humanities*, vol. 7, pp. 1260-1271, 2017.

[59] S. Prathibha, B. Latha, and G. Suamthi, "Particle swarm optimization based workflow scheduling for medical applications in cloud," *Biomedical Research*, pp. 1-1, 2017.

[60] H. Yuan, J. Bi, W. Tan, and B. H. Li, "Temporal task scheduling with constrained service delay for profit maximization in hybrid clouds," *IEEE Transactions on Automation Science and Engineering*, vol. 14, pp. 337-348, 2017.

[61] H. J. Kim, H.-S. Lam, and S. Kang, "Chemical reaction optimization for task scheduling in grid computing," *IEEE Transactions on Parallel and Distributed systems*, vol. 22, pp. 1624-1631, 2011.

[62] J. Xu, A. Y. Lam, and V. O. Li, "Chemical reaction optimization for the grid scheduling problem," in *Communications (ICC), 2010 IEEE International Conference on*, 2010, pp. 1-5.

[63] Y. C. Lee and A. Y. Zomaya, "An artificial immune system for heterogeneous multiprocessor scheduling with task duplication," in *2007 IEEE International Parallel and Distributed Processing Symposium*, 2007, pp. 1-8.

[64] M. Rasti-Barzoki, A. K. Beheshti, and S. R. Hejazi, "Artificial Immune System for Single Machine Scheduling and Batching in a Supply Chain Scheduling Problem," *International Journal of Engineering Science (2008-4870)*, vol. 27, 2016.

[65] V. Vijayakumar, "Trust Based Resource Selection in Grids Using Immune System Inspired Model," in *Proceedings of the 3rd International Symposium on Big Data and Cloud Computing Challenges (ISBCC–16')*, 2016, pp. 213-223.

[66] R. Prodan, M. Wieczorek, and H. M. Fard, "Double auction-based scheduling of scientific applications in distributed grid and cloud environments," *Journal of grid Computing*, vol. 9, pp. 531-548, 2011.

[67] M. D. De Assunçao and R. Buyya, "An evaluation of communication demand of auction protocols in grid environments," in *Proceedings of the 3rd International Workshop on Grid Economics & Business (GECON 2006)*, 2006.

[68] K. Vanmechelen and J. Broeckhove, "A comparative analysis of single-unit vickrey auctions and commodity markets for realizing grid economies with dynamic pricing," in *International Workshop on Grid Economics and Business Models*, 2007, pp. 98-111.

[69] M. Wieczorek, R. Prodan, and T. Fahringer, "Scheduling of scientific workflows in the ASKALON grid environment," *ACM SIGMOD Record*, vol. 34, pp. 56-62, 2005.

[70] J. Taheri, A. Y. Zomaya, P. Bouvry, and S. U. Khan, "Hopfield neural network for simultaneous job scheduling and data replication in grids," *Future Generation Computer Systems*, vol. 29, pp. 1885-1900, 2013.

[71] J. Taheri, A. Y. Zomaya, H. J. Siegel, and Z. Tari, "Pareto frontier for job execution and data transfer time in hybrid clouds," *Future Generation Computer Systems*, vol. 37, pp. 321-334, 2014.

[72] D. de Oliveira, K. A. Ocaña, F. Baião, and M. Mattoso, "A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds," *Journal of grid Computing*, vol. 10, pp. 521-552, 2012.

[73] F. Zhang, J. Cao, K. Li, S. U. Khan, and K. Hwang, "Multi-objective scheduling of many tasks in cloud platforms," *Future Generation Computer Systems*, vol. 37, pp. 309-320, 2014.

[74] R. Achar, P. Thilagam, D. Shwetha, and H. Pooja, "Optimal scheduling of computational task in cloud using Virtual Machine Tree," in *Emerging Applications of Information Technology (EAIT), 2012 Third International Conference on*, 2012, pp. 143-146.

[75] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, and D. H. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, pp. 931-945, 2011.

[76] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, 1998, pp. 69-73.

[77] J. Kennedy, J. F. Kennedy, R. C. Eberhart, and Y. Shi, *Swarm intelligence*: Morgan Kaufmann, 2001.