Developing Data-intensive Scientific Workflow Scheduling Model on Heterogeneous Cloud Computing Resources

Kenesa Bedasa Gutema

A Thesis Submitted to

The Department of Computer Science and Engineering

School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Master's in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

August, 2021

Adama, Ethiopia

Developing Data-intensive Scientific Workflow Scheduling Model on Heterogeneous Cloud Computing Resources

Kenesa Bedasa Gutema

Advisor: Dr.-Ing. Frezewd Lema

A Thesis Submitted to

The Department of Computer Science and Engineering

School of Electrical Engineering and Computing

Office of Graduate Studies

Adama Science and Technology University

August, 2021

Adama, Ethiopia

Declaration

This master's thesis, "Developing Data-intensive Scientific Workflow Scheduling Model on Heterogeneous Cloud Computing Resources" is my original work. That has not been submitted to any university for the award of an academic degree, diploma, or certificate. All sources of information used in this thesis have been properly acknowledged with citations.

Kenesa Bedasa Gutema	_____	_____
Name	Signature	Date

Recommendation

I, the advisor of this thesis, hereby certify that I have read the revised version of the thesis entitled "Developing Data-intensive Scientific Workflow Scheduling Model on Heterogeneous Cloud Computing Resources" prepared under my guidance by Kenesa Bedasa Gutema submitted in partial fulfillment of the requirements for the degree of Mater's of Science in Computer Science and Engineering. Therefore, I recommend the submission of revised version of the thesis to the department following the applicable procedures.

Dr.-Ing. Frezewd Lema	_____	_____
Major Advisor	Signature	Date

Approval Sheet

I, the advisor of the thesis entitled "Developing Data-intensive Scientific Workflow Scheduling Model on Heterogeneous Cloud Computing Resources" and developed by Kenesa Bedasa Gutema, hereby certify that the recommendation and suggestions made by the board of examiners are appropriately incorporated into the final version of the thesis.

Dr.-Ing. Frezewd Lema	_____	_____
Advisor/Supervisor	Signature	Date

We, the undersigned, members of the Board of Examiners of the thesis by Kenesa Bedasa Gutema have read and evaluated the thesis entitled "Developing Data-intensive Scientific Workflow Scheduling Model on Heterogeneous Cloud Computing Resources" and examined the candidate during open defense. This is, therefore, to certify that the thesis is accepted for partial fulfillment of the requirement of the degree of Master of Science in Computer Science and Engineering.

_____	_____	_____
Chairperson	Signature	Date
_____	_____	_____
Internal Examiner	Signature	Date
_____	_____	_____
External Examiner	Signature	Date

Finally, approval and acceptance of the thesis is contingent upon submission of its final copy to the Office of Postgraduate Studies (OPGS) through the Department Graduate Council (DGC) and School Graduate Committee (SGC).

_____	_____	_____
Department Head	Signature	Date
_____	_____	_____
School Dean	Signature	Date
_____	_____	_____
Office of Postgraduate Studies, Dean	Signature	Date

Acknowledgment

First and foremost, praises and thanks to the Lord, the Almighty, for His showers of blessings throughout my research work to complete my research successfully.

I would like to express my deep and sincere gratitude to my instructor and research advisor, Dr.-Ing. Frezewd Lema. Words fall short to thank him, for his invaluable guidance and support during my studies and for his careful instruction during these 2+ years. He has taught me the methodology to carry out the research and to present the research works as clearly as possible. It was a great privilege and honor to work and study under his guidance. I am extremely grateful for what he has offered me.

I would also like to thank my classmates and research colleagues for their constant encouragement, invaluable feedback and suggestions. Without their support, I could not have undertaken my Masters. I am also very grateful to Ambo University and Adama Science and Technology University for this invaluable opportunity. I would like to offer my special thanks to my friends for genuinely supporting, encouraging, and motivating me.

I am extending my heartfelt thanks to those friends I met in ASTU. Graduate students in the Department of Computer Science and other. I was never alone in my study with these nice people around. I have learnt a lot of things from them: friendship, empathy and great sense of humor.

Finally, I would like to thank my family for their love, prayers, support, patience and sacrifices for educating and preparing me for my future. Thank you, Dad and Mom for standing by my side every time, everywhere.

Thank you!

Dedicated

To

Dad and Mom.

Table of Contents

List of Figures

List of Tables

Lists of Acronym

ANN	Artificial Neural Network
BW	Bandwidth
CB	Cloud Broker
CIS	Cloud Information Service
CP	Critical path
CPU	Central Processing Unit
DAG	Directed Acyclic Graph
DC	Data Center
GB	Giga Byte
GUI	Graphical User Interface
HEFT	Heterogeneous Earliest Finish Time
IaaS	Infrastructure-as-a-Service
LIGO	Laser Interferometer Gravitational Wave Observatory
MB	Mega Byte
Mbps	Megabits per second
MIPS	Millions of instructions per second
NP	Non Polynomial
PaaS	Platform-as-a-Service
PE	Processing Elements
PSHA	Probabilistic Seismic Hazard Analysis
QoS	Quality of Services
RAM	Random Access Memory
SaaS	Software-as-a-Service
SIPHT	sRNA Identification Protocol using High throughput Technology
SLA	Service Level Agreement
SP	Scheduling Plan
sRNA	Small Ribonucleic Acid
SWf	Scientific Workflow
SWfMS	Scientific Workflow Management System
TB	Terabyte
VM	Virtual Machine

Abstract

Cloud computing is a distributed and powerful IT architecture that allows cloud providers to efficiently supply computing services to users on demand. Cloud computing is quickly becoming the preferred method for large-scale and complicated computation. For the reason of the shortage of suitable computing facilities on local servers and the growing volume of data produced and consumed by the experiments, a number of scientific experiments are being moved to cloud. Several existing large-scale scientific workflows based on simulations produce very large files and datasets and require huge amounts of computing resources to execute. Cybershake workflow, for example, process over 200 TB of data file which is generated during execution. Montage workflow creates data files that are several GB in size, but only 30% of CPU time is assigned to computation and the remaining 70% is used by I/O operations. Consequently, it should be addressed by eliminating unnecessary data movement or minimizing the probability of transferring data as much as possible, resulting in a shorter total workflow execution time. This proposed work focuses on tackling both data file transfer problem that is enormously time-consuming task in workflow execution and scheduling of tasks within workflow. A score-based data-intensive scientific workflow scheduling algorithm on heterogeneous cloud resources that allocates resources to workflow application tasks based on the performance of VMs like processor speed, processing cores, storage capacity (secondary storage), bandwidth strength, and RAM size. For the evaluation of our work, we used a WorkflowSim 1.0, which is an extended simulation toolkit from CloudSim for workflow simulation. We compared our scheduling algorithm to three standard scheduling algorithms: Round Robin, MinMin and MaxMin scheduling algorithms. In terms of Makespan, the proposed scheduling method outperforms all three heuristics.

Keywords: Data-intensive, Cloud Computing, Scientific Workflow, Workflow Scheduling, Heterogeneous Resource

CHAPTER ONE

INTRODUCTION

1.1 Background of the Study

We are entering a new era of computing in today's implausibly sophisticated world of computational power, complex data storage, high-speed processing capabilities, and next-generation telecommunications. Computations can now take place anywhere on the planet, and information or resources (i.e., hardware, software, apps, operating system, etc.) can be accessed via networks from anywhere. Computing as a utility has long been a dream of the computing industry, allowing services to be used and accessed from any location. As a result, IT services and applications can be delivered via the Internet. With the introduction of Cloud Computing technology, however, this dream became a reality. Access to a lot of computer power in a completely virtualized way over a distributed network is now accessible with this new computing paradigm, Cloud Computing.

Cloud computing is a relatively new computing technology that has likely been one of the most significant technological breakthroughs in the last decade. Cloud computing is an internet-based service that offers dynamically scalable computing infrastructures, as well as manageable, controllable, and schedulable virtual services such as servers, storage space, computing services, networks, and so on, presented as one or more unified computing resources on-demand to the user (Groom, 2018; Kale, 2017; Marston et al., 2011). It provides services to clients through a utility-oriented approach, i.e., in the same way that electricity, telephone, gas, and other utilities provide services to customers through the internet using a pay-per-use model. Computational resources in Cloud Computing are provided to customers as a form of endless resource pool (e.g. processing capacity, memory, storage, etc.), thanks to Virtualization technology. For its consumers, this makes it a more convenient platform than traditional hosting and computing services.

Cloud computing is a distributed and powerful IT architecture that allows cloud providers to efficiently supply computing services to users on demand. Cloud computing is quickly becoming the preferred method for large-scale and complicated computation. Virtualization technology is a technique for dividing physical infrastructure into “virtual” devices or environments that may be used more effectively and efficiently. Virtualization in cloud

computing is designed in order to increase capacity utilization, efficiency and scalability (Hashem et al., 2015). Allowing multiple customers and enterprise to share applications, is one of the most essential features of virtualization. It allows customers to utilize resources while reducing the number of physical systems required.

Virtualization technology is also important in cloud computing because it allows users to share resources with predefined QoS constraints. Customers can get services from the cloud provider with specific QoS constraints. Both the Cloud Provider and the Cloud Consumer agreed on the minimum level of services that should be maintained under a Service Level Agreement (SLA). To avoid being fined, Cloud Providers should supply services to clients with specific QoS constraints. Cloud users must obtain assurances from suppliers regarding service provisions (Chana & Singh, 2014). Storage, computation, networking hardware, and software services are examples of these services. Both efficient provisioning and scheduling are part of manipulation of these resources. The procedure of detecting, selecting, and allocating suitable resources required to complete tasks and workflows is referred to as resource provisioning, while scheduling in cloud refers to the process of allocating jobs to virtual machines (VMs) or allocating virtual machines to available resources to meet customer requirements (Chenni Kumaran & Aramudhan, 2018; Vijaya & Srinivasan, 2016). Cloud resource management must consider from two points of view: from the cloud customer and cloud provider point of view. Clients want high-quality service at reasonable pricing, whereas cloud providers concentrate on providing QoS while earning a profit (Luiz F. Bittencourt et al., 2018).

One of the goals of scheduling is to identify the optimal resource for completing execution of tasks. So that the scheduling algorithm can improve various quality of service (QoS) parameters such as resource utilization, task rejection ratio, reliability, power consumption, execution cost, and so on, without affecting the service level agreement (SLA), while also taking into account constraints (deadlines, priorities, and so on) and avoiding the load imbalance (overused and underutilized) problems (M. Kumar et al., 2019). The multi-tenant, on-demand, elastic, and pay-as-you-go resource model, which provides heterogeneous types of resources, causes one of several scheduling issues (Rodriguez & Buyya, 2017). In order to meet a user-defined deadline, an optimal resource scheduling strategy should avoid resource contention, resource fragmentation, over-provisioning, and under-provisioning of resources (Chenni Kumaran & Aramudhan, 2018). It should also meet a user-defined deadline, minimize

computational cost, minimize makespan, and maximize the success rates of the proposed algorithm with respect to user limitations.

A number of business organizations and individuals have turned to cloud computing to store, process, and analyze large amounts of data. For the reason of the shortage of suitable computing facilities on local servers and the growing volume of data produced and consumed by the experiments, a number of scientific experiments are being moved to cloud. To support users in accessing cloud resources and launching their programs, cloud service providers have begun to implement parallel data processing frameworks into their services (Hashem et al., 2015). Scientific computing applications and platforms are evolving from CPU-intensive tasks executed over strongly coupled infrastructures, i.e., complicated simulations run on supercomputers, to data-intensive systems requiring flexible computing and storage resources based on the needs and budgets of users (Marozzo et al., 2017). Scientific experimenters in areas such as earthquake science, bioinformatics, geophysics, energy physics, astronomy, weather monitoring, and gravitational wave physics are using cloud environments to share, manage, and process large data sets because it requires a large number of complex computations and a large data transfer (G. Kaur & Mathai, 2018).

These scientific applications are modeled as workflows with multiple dependent tasks and are called scientific workflow that is represented as a directed acyclic graph (DAG). The edges describe data and control dependencies between the nodes, which represent computations. By adjusting the input parameter or dataset, this workflow can be run as many times as needed to check and analyze the values of the output (Chen et al., 2015; Teylo et al., 2017). Simulation, data analysis, image processing and many other tasks are covered by these scientific workflows. Examples of Scientific Workflow include Montage, Epigenomics, CyberShake, SIPHT, and LIGO are familiar ones. Epigenomics: DNA grouping information acquired from the hereditary investigation process is part of a few lumps and is used to delineate the epigenetic condition of human cells. LIGO: identifies gravitational influxes of vast starting points by watching stars and dark openings. Montage: makes a mosaic of the sky from a few information pictures. CyberShake: utilizes the Probabilistic Seismic Danger Investigation (PSHA) strategy to portray earth-shudder risks in the district. SIPHT: scans for little un-deciphered RNAs encoding qualities for the entirety of the bacterial imitations in the National Center for Biotechnology Information (NCBI) database (Juve & Deelman, 2010), (Juve et al., 2013).

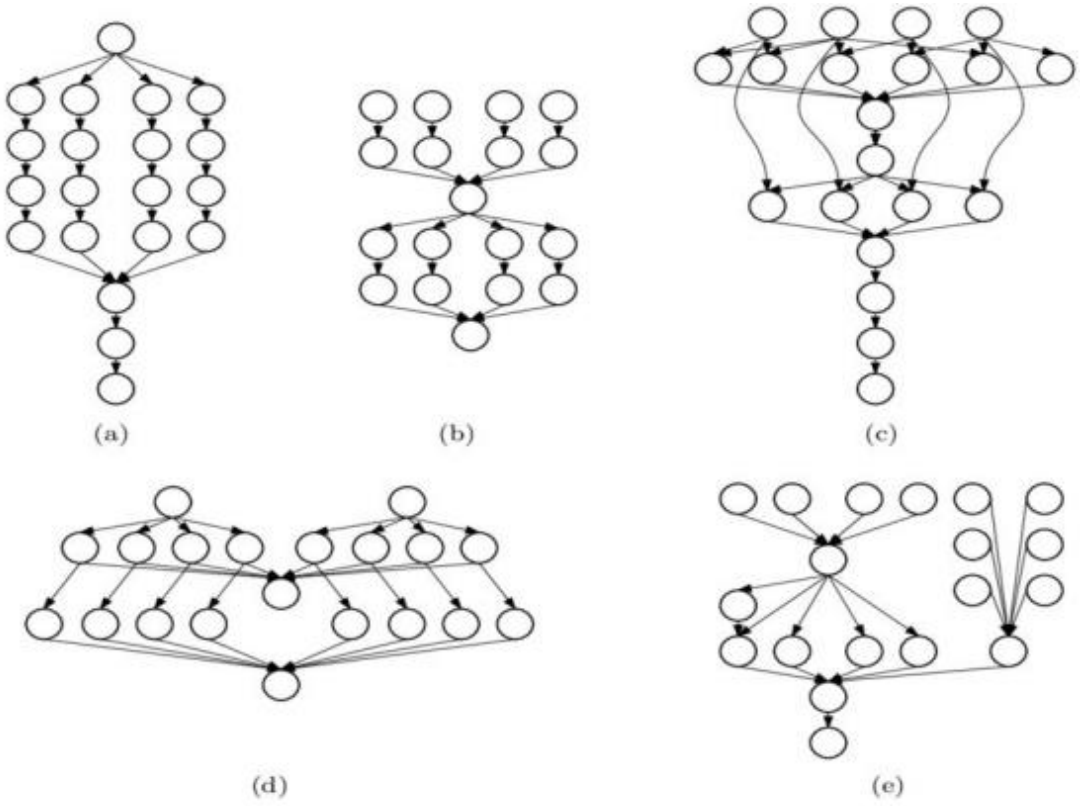


Figure 1. 1 Scientific Workflow examples [(a) Epigenomics, (b) LIGO, (c) Montage, (d) CyberShake, (e) SIPHT](Juve et al., 2013).

Data-intensive computation involves manipulating and producing massive amounts of data, ranging from megabytes to petabytes, as well as meeting runtime requirements. Aside from job scheduling, the execution of such workflows demands a careful selection of data management solutions, as data transfers can have a major impact on the execution time (Sukhoroslov, 2019). The appropriate assignment of a virtual machine to tasks is required for the execution of such data- and computation-intensive workflows. As a result, assigning these virtual machines and deciding where to execute activities requires an optimum workflow scheduling algorithm, which is a popular issue among researchers right now. During execution, such a workflow consumes and produces a very large size of data (many GBs or PBs of data) that can be then used as an input for other intermediate tasks. Therefore, these data should be transferred from the source virtual machine to the destination virtual machine, where tasks that require the data as an input exists for execution.

Moving large-size data (maybe in GBs or PBs) between different virtual machines several times during the execution of workflow can have a lot of influence over the execution time of the workflow. Therefore, avoiding such an unnecessary data transfer between virtual machines, or minimize the chance of moving data as much as possible in a way it does not affect workflow execution, and focusing on minimization of execution time is very important. Identifying the critical tasks to be scheduled at each phase for a specific VM is a difficult problem to solve. The primary reason for the scheduling length may be due to the fact that critical tasks have been delayed. This is the focal point of this paper. This proposed algorithm finds the cloud resources with high profile i.e., resource with the highest computational capabilities and map it to the high priority task. Based on this assignment the successor task can also be mapped to the same virtual machine with its predecessor task. So that, unnecessary data movement can be avoided. This can lead to the completion of workflow execution within a short period. By task priority, we mean the importance of thinking about the task in the first place. The importance of a task is determined by a task priority factor. As such, high-priority tasks should be finished first, whereas low-priority ones should be handled after higher-priority tasks have been completed.

1.2 Motivation

Scientists began shifting their scientific experiments to clouds in the previous decade, where computing resources (infrastructure, platform, and software) are provided on-demand over the Internet to clients. Motivated by this scenario, we decided to address task scheduling issues using cloud systems' strengths and potential. Today, big scientific breakthroughs have allowed us to have a better understanding of our environment. The majority of these experiments are simulations based on a complicated method that analyzes vast amounts of data acquired in a computationally intensive setting. Scientific and other real-world scenarios are increasingly using data-intensive services. Computationally intensive and large-scale data processing applications are modeled using scientific workflows. Within each transaction, which retrieves data from an instrument or a database, reformats the data, and runs an analysis, these data-intensive workloads require access to numerous datasets. By scaling up or down highly accessible resources, this data-intensive procedure may take advantage of cloud dynamism. Using clouds for scientific workflow saves a lot of money because it eliminates the need for expensive infrastructure and the labor of setting up new infrastructure (Teylo et al., 2017).

The fundamental purpose of running a data-intensive Scientific Workflow on a distributed environment like the cloud is to minimize job execution time. However, makespan of the workflow is dependent on data and file interactions with tasks. The execution time of the workflow as a whole is affected by these stage-in and stage-out data files. As a result, scheduling workflows with proper file management can cut down on the total time it takes to complete an execution of tasks and the expense of doing so. It is feasible to eliminate high bandwidth utilization costs and significantly minimize completion time of workflow by preventing unnecessary file movement from one virtual machine to another virtual machine.

Resource allocation is concerned with maximizing resource use and allocating limited resources among many tasks to be completed. The distribution of resources has a significant impact on task performance, system availability, and responsiveness. In workflow, which consists of a high number of interdependent jobs, the optimal scheduling method is especially important. Interdependent tasks communicate via sharing data files, which must be available during their execution. Transferring significant amounts of data to the successor task uses a lot of bandwidth and wastes a lot of CPU time.

1.3 Statement of the Problem

Users should be able to share the limited resource efficiently, which benefits both cloud providers and consumers. Users have the ability to request a large number of resources at once. By its very nature, user demand is dynamic. The allocation of resources to these user demands must also be dynamic. Task scheduling on the server is tough due to the enormous number of task requests that demand certain resources to execute. Cloud Resource Scheduling is a critical component of cloud computing resource management (Rana & Tanwar, 2014). As a result, the server's schedule must be optimal and sufficient to ensure that each user request receives a timely response and that each task receives the resources required to finish it (Aazam & Huh, 2014; Jagannatha, 2019; Rana & Tanwar, 2014). Task scheduling is a big issue in the cloud, as it causes resource allocation to perform poorly in a dynamic context. This complicates sending the task to resources from the queue, and it also affects providing or allocating the task to the cloud data center.

Several existing large-scale scientific experiments based on simulations produce very large files and datasets and require huge amounts of computing resources to execute. A single workflow

generally consists of a set of tasks each of which may communicate with another task in the workflow. Since tasks are typically a sequential (single node) application, a data transfer among tasks generally handled by the workflow system (de Oliveira et al., 2019; Shibata et al., 2010). So data files may be transferred from node to node explicitly or implicitly. Since there are inter-dependencies among tasks of workflow, it is difficult to attain the required level of resource utilization. Taking data transfer overheads into account, is critical in the case of data-intensive computations. For example, using one not powerful resource for two connected tasks (no transfer overheads) may be more efficient than using two powerful separated resources due to significant data transfer time. A distributed computational environment consists of heterogeneous resources with differences in performance, network bandwidth, access policies, etc. (Nasonov et al., 2017). Provided that, scheduling Scientific Workflow with the data-intensive application is appearing a vital research issue.

To be processed in an acceptable amount of time, a large-scale data workflow must be run in parallel in a distributed environment, such as a cloud platform with high processing capacity (de Oliveira et al., 2019). Data must be transferred and ready for execution before tasks may begin to be executed. G. Juve et al. (Juve & Deelman, 2010) revealed that an astronomy application of Montage workflow contains 10,429 tasks, reads 4.2 GB of input data, and produces 7.9 GB of output data, whereas a seismology application (Broadband) contains 320 tasks, reads 6 GB of input data, and writes 160 MB of output data in their case study on Amazon's EC2 cloud. Epigenome workflow is a bioinformatics application with 81 jobs that reads 1.8 GB of input data and generates 300 MB of output data. Scientific Workflow provides the time required to complete each task of the workflow, as well as the total number of files produced and consumed during the workflow's execution. Data files range in size from 2 GB (SIPHT process) to over 200 TB (Cybershake workflow) (Teylo et al., 2017). Montage workflow, for example, creates data files that are several GB in size, but only 30% of CPU time is assigned to computation (the remaining 70% is used by I/O operations) (Juve et al., 2013).

Shibata et al. (Shibata et al., 2010) also studied the Montage workflow and found that I/O operations take up 99% of the CPU time (in some cases). As a result, a considerable portion of the total workflow execution time will be spent on data transmission rather than data processing while transferring data files from one virtual machine to another virtual machine. Consequently, it should be addressed by eliminating unnecessary data movement or minimizing the probability

of transferring data as much as possible in a way that does not interfere with workflow execution, resulting in a shorter total workflow execution time. However, workflow tasks must be assigned to the most appropriate virtual machines in order to reduce the overall workflow execution time. Each task must be scheduled to a corresponding VM in order to execute a data- and compute-intensive workflow in the cloud. The scheduling problem then becomes deciding where all jobs should be completed. Even in basic cases, assigning jobs to distributed computing resources is an NP-complete issue (de Oliveira et al., 2019).

However, there are some aspects of cloud computing that make the scheduling procedure a little more difficult. Resources in cloud environments can be of different types and generations. There is a possibility that this will affect the performance of virtual resources in a given virtual machine. Furthermore, the disparity in hardware can adversely affect performance (Negru et al., 2017). Several VMs in cloud is deployed with different performance metrics; each of which has its own processing and storage capacity, as well as various bandwidth and financial costs. Furthermore, some of these VMs, such as the Amazon EC2 micro and nano VMs, may not be ideal for High-Performance Computing (HPC) (Teylo et al., 2017). As a result, when running workflows in parallel in the cloud, the deployed virtual cluster is frequently made up of heterogeneous VMs, and this heterogeneity must be taken into account while scheduling. Many heuristics, such as Partitioned balanced time scheduling (PBTS), MinMin, MaxMin, Critical-Path-on-a-Processor (CPOP), Heterogeneous-Earliest-Finish-Time (HEFT), are used to solve task scheduling challenges in the cloud. Particle Swarm Optimization (PSO), Symbiotic Organism Search (SOS), Ant Colony Optimization (ACO), Bees-Inspired Algorithm (BA), Genetic Algorithm (GA), and a slew of other meta-heuristics have been developed. However, throughout the scheduling process, these algorithms do not take data files into account.

1.4 Research Question

a) How the minimization (if possible, avoidance) of the movement of data files within the Scientific Workflows does affect the scheduling makespan of Scientific Workflow on Cloud?
b) How does the optimal selection of parallel tasks within the Scientific Workflow for mapping onto high performance cloud resource affects the execution time of scheduling?
c) How VMs are selected from the pool of resources to host tasks?

1.5 Objectives of the Study

1.5.1 General Objective

The general objective of this thesis is to design an optimal scheduling algorithm for data-intensive Scientific Workflow, which can reduce the makespan of the execution of tasks.

1.5.2 Specific Objective

a) To identify different performance metrics for cloud VMs and collect scientific workflows/tasks
b) To design, and implement a new novel algorithm solution for data-intensive Scientific Workflows scheduling that tries to avoid (in case it is possible) or minimize unnecessary data transfer in between tasks
c) To defines the efficient way which tasks are selected for assignment onto virtual machines
d) To conduct the experiment the accuracy of the proposed algorithm using synthetic compute- and data-intensive scientific workflow applications to evaluate the performance of the algorithm.
e) To experimentally evaluate the proposed algorithm with improvements over traditional scientific workflow scheduling algorithms; explain it in terms of evaluation metrics.

1.6 Scope and Limitations

The proposed algorithm will focus on scheduling data-intensive scientific workflow, which consists of multiple interdependent tasks on cloud environment. It helps Scientific Workflow that works with massive data files to finish execution in a short period. This study assumes that all necessary virtual machine for executions of workload is already provided dynamically based on the metrics of QoS and it schedules only on the reserved resources and does not give much emphasis on creating virtual machine scheduling policy. Therefore, resource provisioning is out of the scope of this work and can be added in the future. Besides that, this research work does not address issues related to power consumption of VMs, which is strongly related to resource scheduling in cloud, due to the lack of time.

1.7 Application of Results

Scientists need their experiment to execute in a short period whether it is on a cluster, grid or cloud. This proposed approach is applicable for cloud-based Scientific Workflow to schedule

tasks in a way it does not have to consume much time transferring data in between different tasks. Its focal point is the heterogeneity of resources in cloud. Data-intensive Scientific Workflow needs to be scheduled to resources that process it in a short period. This algorithm is proposed for workloads that require enormous data files to process in the cloud environment. Therefore, the result of this work is could improve the performance and resource utilization in the cloud environment.

1.8 Beneficiaries of the study

The beneficiaries of the study are given below:

1.8.1 Cloud End-users

Cloud end users are those who can access various cloud resources through the internet for getting cloud services and pay for the use. In our case, cloud end-users may include scientists, academicians, researchers, and scientific organizations involving in scientific workflow execution. A good scheduling algorithm will satisfy user requirements by improving the execution time (makespan) and minimizing the waiting time of tasks.

1.8.2 Cloud Service Providers

Cloud providers can be benefited from a good scheduling algorithm for it can satisfy users' requirements with minimum cloud resources. It enables to easily management a cloud resource and they become more cost-effective due to optimal scheduling of a task and balancing the entire load of the system. Therefore, optimal scheduling algorithm saves resource and energy consumption for renting to other users simultaneously.

1.8.3 Researchers and Academicians

Researchers and academicians for future study of further optimization can use the proposed algorithm. It can be used as a reference at the university level for academic and research purposes.

CHAPTER TWO

LITERATURE REVIEW

2.1 Overview of Cloud Computing

Cloud computing is a technology that is now gaining a lot of attention and being widely employed. It has evolved into a viable method for service providers to offer enterprise applications as well as expand or introduce new services. In a nutshell, Cloud Computing is an internet-based service that offers users on-demand access to dynamically scalable computing infrastructures, manageable, controllable, and schedulable virtual services such as servers, storage space, computing services, networks, and so on, presented as one or more unified computing resources on a pay-as-you-go basis (Aazam & Huh, 2014). The economic model of the cloud versus the grid is one of the most fundamental distinctions between the two. The cloud uses a pay-as-you-go strategy, whereas the grid uses resource exchange. Clouds also help with pricing, accounting, SLA management, visualization, and security control, among other things (de Oliveira et al., 2019).

Cloud computing provides computing, storage, and network resources through infrastructure, platform, and software services, with the illusion that resources are unlimited. Although it is frequently confusing, five fundamental differences exist between cloud and grid computing. The first is that the cloud leverages virtualization techniques to offer scalable, physical infrastructure-independent services (Brandic & Dustdar, 2011). The second is that it delivers not only infrastructure services, such as processing or storage resources, but also software and platform services. In the cloud, a cluster made of VMs can be configured and used. In addition, database management systems are available as a cloud platform. The third distinction is the dynamic supply potential. A list of resources is usually set during the complete workflow execution in a grid environment. Thus, dynamic supply in the grid is quite unusual, as it does not offer any advantage.

In cloud environments, by contrast, we have a resource type list from which a theoretically endless number of resource instances can be provided. This dynamic provisioning can deliver even more advantages, including improved performance and lower financial costs. The fourth difference is that the service level agreement (SLA) is used to define the quality of service offered by service providers to the users. Cloud SLA contains relatively complete on-demand

performance metrics whereas grid SLA is relatively informal (Brandic & Dustdar, 2011; J. Liu et al., 2015). Fifthly, the cloud supports pricing and accounting services, which are not necessary for the grid. Certain grids are changing to the cloud. For example, Grid'5000 is a French grid, which provides virtualized big data (e.g. Hadoop) resources and services.

In some ways, cloud computing resembles parallel computing such as grid computing and cluster computing, but it manages its resources using virtualization techniques. This distinguishes it from parallel computing(Brandic & Dustdar, 2011), (Zhan et al., 2015). But there's more: heterogeneity, elasticity, billing, and resource pooling are all crucial differences (Vijaya & Srinivasan, 2016). Virtualization is the basic technology and component that enables Cloud Computing to give a virtual environment to the end-user, which might be software or hardware. Storage, network, desktop, application, server, and other services provided by Cloud Providers are all virtualized environments. Because of virtualization, physical resources can now be abstracted, allowing multiple applications to share a single physical resource and a single physical instance of a resource or application to serve multiple customers (Vaezi & Zhang, 2017). Not only may computational resources be virtualized, but storage resources can also be aggregated and presented to cloud users to increase the capability of storage services. Networks, servers, and desktops can all be virtualized in addition to storage and computational resources.

This virtualization strategy provides significant technical benefits to cloud computing (Shea et al., 2014). It encompasses things like availability, mobility, usage, machine cost and space savings, and energy consumption reduction, among other things (Vaezi & Zhang, 2017). It also aids in the cloud-based IT structure's simplification and accurate charging of end-user service use. Virtualization also helps in the effective management of underlying hardware resources, and once this is completed, a scheduler for utilizing underlying hardware resources is implemented (Mathew, 2018). To get the most out of these resources, you need to find out the best job execution sequences from users and map them to physical resources. As a result, resolving the assignment of these virtual machines and deciding where to perform those activities is not an easy operation. Another advantage of virtualization is the ability to migrate virtual machines from one device to another in the event of hardware failure or update (Vaezi & Zhang, 2017).

2.2 Essential Characteristics of Cloud Computing

Cloud takes care of the essential infrastructure, allowing staff to focus on innovation rather than infrastructure concerns. NIST has enlightened every cloud to incorporate five key characteristics of Cloud computing (Peter Mell (NIST); Tim Grance (NIST), n.d.).

2.2.1 On-demand Self-service

The ability of end-users to self-provision resources in a computing environment other than the cloud fundamentally disturbs most legacy procedures such as capacity planning, network management, and security. The other side effect of the aforementioned circumstance is extreme inefficiency and resource waste, as it allows unnecessary resources to be given at one end while others go starving. Cloud-based architectures, on the other hand, are intended and constructed to support self-provisioning. Without requiring human interaction with each service provider, a consumer can unilaterally provision computer capabilities, such as server time and network storage, as needed (Jagirdar et al., 2013). On-demand resource allocation is a feature of cloud platforms. Users of the cloud can request and expect adequate resources for their needs at any time. The "illusion of unlimited resources" has been coined to describe this aspect of clouds (Juve & Deelman, 2010).

The disadvantage of this method is that, unlike best-effort queuing, it does not allow for waiting time. If adequate resources are not immediately available to service a request, the request will fail. On-demand provisioning enables workflows to be more flexible in their resource selection. Unlike closely connected programs that require all resources up front and prefer to queue to ensure priority, a workflow application might begin with only a portion of the total resources required. For workflows with just serial tasks, the minimum viable resource pool is one processor. With on-demand provisioning, a workflow can allocate as many resources as it wants and get right to work (Juve & Deelman, 2010).

2.2.2 Broad Network Access

Computing resources were limited and expensive in the beginning eras of computing. Likewise, network resources were limited. Costs connected with the network have decreased over time as a result of manufacturing scalability, commoditization of associated technologies, and market rivalry (Jagirdar et al., 2013). As network bandwidth expanded, so did network access and scalability, which are key enablers for cloud computing. Cloud capabilities are accessible

through the internet and through standard procedures, which are then used by a variety of thin and thick client platforms, including smartphones, laptops, and personal data assistants.

2.2.3 Resource Pooling

Using a multi-tenant approach, the provider's computing resources are pooled to serve numerous clients, with distinct physical and virtual resources dynamically assigned and reassigned based on consumer demand. Storage, processing, memory, network bandwidth, and virtual machines are pooled by the cloud provider, and these resources are dynamically assigned and reassigned based on customer demand. The subscriber has no control or knowledge of the precise location of the given resources, but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter) (Jagirdar et al., 2013; Rajkumar Buyya, Christian Vecchiola, 20143). Examples of resources include storage, processing, memory, network bandwidth, and virtual machines.

2.2.4 Rapid Elasticity

It refers to a system's ability to adapt to workload changes by autonomically providing and de-provisioning resources such that available resources match current demand as nearly as possible at any given time. It is built on dynamic infrastructure, and a standardized, scalable, and secure physical infrastructure is frequently the foundation for the dynamic infrastructure. There is usually a high amount of redundancy to provide high availability, but it is usually designed in such a way that it is simple to expand as demand grows, without needing architecture redesign. Capabilities can be provisioned and released fast and elastically, in certain circumstances automatically, to swiftly scale out and quickly scale in. The capabilities offered for provisioning appear to be limitless to the client and can be acquired in any number at any moment (Jagirdar et al., 2013). Clouds allow users to return resources on demand in addition to supplying them on demand. This dual capacity, known as elasticity, is a very beneficial characteristic for workflow applications because it allows workflow systems to rapidly expand and contract the available resource pool as the workflow's needs alter over time (Juve & Deelman, 2010). Because of these changes, it may be more cost-effective to purchase or release resources to better match the application's requirements and ensure that resources are effectively utilized.

2.2.5 Measured Service

Cloud systems utilize a metering capability at a level of abstraction suited to the type of service to automatically control and optimize resource use (e.g., storage, processing, bandwidth, and active user accounts). Resource utilization can be tracked, regulated, and reported, giving both the provider and the user of the service complete transparency. Consumers pay for only the resources they use in cloud computing, and are therefore invoiced or billed on a consumption-based model. Cloud computing platforms include tools for capturing consumption data, which can be used for chargeback reporting and/or billing system integration.

2.3 Cloud Service Models

There are different level of abstraction as we move up or down the stack of cloud. It includes Software-as-a-service (SaaS), Platform-as-a-service (PaaS) and Infrastructure-as-a-service (IaaS).

SaaS - In this paradigm, the end-user receives both a ready-made application and resources as a demand-based service. Multiple end-users can access a single instance of such a service through the internet using web browsers or other apps such as VDI (Virtual Desktop Infrastructure) or mobile applications (Groom, 2018; Mathew, 2018). Subscribers do not need to download a copy of the application on their computers; instead, they can use SaaS providers to access it.

PaaS - It is a computer platform or development environment that is provided as a service to subscribers, allowing them to develop and run programs via the internet. End users can design and execute their own apps with the support of the provider's infrastructure (Groom, 2018; Kale, 2017; Mathew, 2018). This development environment has the ability to scale up and down as needed.

IaaS - Infrastructure as a Service is a fundamental layer in the Cloud Computing architecture. IaaS provides end-users with large computational resources and infinite storage space as a pay-per-use service (Groom, 2018; Mathew, 2018). Subscribers can rent servers, data center space, and network equipment as a service and build it together as needed rather than purchasing it (Kale, 2017; Vaezi & Zhang, 2017).

2.4 Model for Cloud Deployment

There are different types of cloud providers from which any cloud user can choose to receive service. For their businesses, different organizations or users prefer different deployment methodologies. These deployment strategies are:

Public Cloud - A public cloud is one that may be accessed by anybody with an internet connection and a third-party provider. Despite the security concerns, using the public cloud does not mean that users' data is accessible to the whole public. It is available to anyone who is interested in subscribing at a low cost and deploying it in an elastic fashion with some control mechanisms.

Private Cloud - As the name implies, a single business can access services or infrastructures via private networks. Private cloud, in contrast to public cloud, develops a unique cloud platform that is also more secure and controllable. The services are hosted in a secure data center and are only available to corporate personnel (Mathew, 2018).

Hybrid Cloud - This is a hybrid cloud that combines private and public cloud services. Hybrid cloud is a sort of cloud that allows data or applications to flow between the two clouds, allowing customers to outsource non-essential data to the public cloud while maintaining control over the critical data. It is ideal for businesses that can benefit from both cloud services at the same time (Vaezi & Zhang, 2017).

Community Cloud - A cloud deployment architecture in which enterprises pool infrastructure to serve specific communities with shared concerns (Groom, 2018). A cloud environment designed for collaborative work by businesses or users (Vijaya & Srinivasan, 2016).

2.5 System Architecture in Cloud Computing

In the Infrastructure as a Service (IaaS) cloud architecture, the cloud provider handles a vast set of computing resources, such as processing capacity. In other words, it is the distribution of hardware like servers, storage, and networks, as well as associated software (operating system and file system), as a service. In the IaaS cloud, the cloud provider does very little management other from keeping the data centers up and operating. The cloud user can deploy, configure, and administer software services just like if they owned the hardware (Mimura Gonzalez et al., 2017). The other architectural layers of cloud computing are Platform as a Service (PaaS) and

Software as a Service (SaaS). Platform as a Service refers to when a cloud provider delivers both the hardware and a specific quantity of application software. SaaS (Software as a Service) is a hosted set of software that you only pay for when you use it. It can be thought of as a local application runtime replacement (S., 2011).

In a cloud computing architecture, a front-end and a back-end are connected over the Internet. The interface includes client devices, which might be thin clients, thick clients, or mobile devices. Customers must use an interface and programs to access the cloud computing system. The back-end is made up of several servers and data storages. In most cases, a central server controls the cloud system, keeps track of all traffic, and satisfies customer demands efficiently. Cloud computing is a new type of distributed computing that includes infrastructure, platform, and software as a service.

2.6 Key Enabling Technologies of Cloud Computing

The concept of renting computing services using enormous distributed computing facilities has been around for a long time, dating back to the early 1950s mainframes. Technology has progressed and improved since then. This technique has produced a number of favorable conditions for cloud computing adoption (Rajkumar Buyya, Christian Vecchiola, 20143). Virtualization, distributed systems, and Web 2.0 technologies, service orientation, and utility computing are examples of the evolution of distributed computing platforms that have propelled cloud computing forward (S., 2011). The introduction of the Cloud computing concept relied heavily on these five basic technologies. Cloud computing encourages enterprises to use these technologies by removing the requirement for extensive knowledge and experience with them.

2.7 Resource Management in Cloud Computing

Ever increasing demand in Cloud Computing realm has brought forth a more complex computing environment with heterogeneous computational resources. Clouds have already demonstrated their suitability to execute workflows that demand HPC (Aazam & Huh, 2014). Thus, to execute a workflow in the cloud, all activations have to be scheduled and executed in parallel on a set of VMs: $VM = \{vm1, vm2, \ldots vmd\}$ (de Oliveira et al., 2019). When we talk about resources, we mean any entity that is designed to serve end-users at any given time. It can be physical resources (storage, memory, CPU, networking, etc.) or logical resources (throughput, bandwidth, operating system, etc.).

There are many different types of resources. The resources involved in this paper include - *Data Center (DC):* datacenters are comprised of physical machines and virtual resources. *Physical Machines (PMs):* are the compositions of physical computing devices in a cloud data center; each PM can host multiple virtual machines, and can have more than one CPU, memory, hard drive, and network cards. The failure of any physical nodes may disable some VMs installed on the failing nodes. However, the failure of VMs will not pull down the host system. The physical machines are also called host systems. *Physical Clusters:* consist of a number of PMs, necessary networks, and storage facilities.

Virtual Machines (VMs): A virtual machine (VM) is a computing engine with predefined virtual resources such processors, memory, and storage, as well as a computing stack that includes an operating system, middleware, and one or more application programs (Ismaeel et al., 2016). Virtualization creates an abstraction layer over physical hardware using software and accomplishes this by establishing a virtual computation system, virtual machines (VMs). This enables businesses to operate several virtual machines, operating systems, and applications on a single physical server, thereby dividing it into multiple virtual servers. Virtual machines (VMs) are constructed on PMs using virtualization software; each VM can have many virtual CPUs, hard disks, and network cards. A virtual machine runs on a guest operating system, which is often different from the host operating system. The resources in the physical machine where the VM is installed are managed by the Host OS. Each virtual machine (VM) can be deployed on a remote server or replicated over multiple servers in the same or other physical clusters.

Virtual Clusters: To complete tasks, scientific workflows require a huge number of computation cycles. Virtual machines supply these cycles in the cloud. Many virtual machine instances must be used simultaneously to obtain the performance required for large-scale processes. A cloud provider's computing resource is referred to as an instance. Virtual clusters are groups of virtual machines (Juve & Deelman, 2010). Contextualization is the process of deploying and configuring a virtual cluster. Contextualization involves a series of complicated configuration procedures that can be time-consuming and error-prone when done manually. Software like the Nimbus Context Broker can help automate this procedure (Juve & Deelman, 2010; Keahey & Freeman, 2008). This application collects data about the virtual cluster and utilizes it to create configuration files and launch services on cluster VMs. It consists of a number of virtual machines (VMs), as well as the necessary networks and storage.

VMs are installed on distributed servers from one or more physical clusters to create virtual clusters. A virtual network interconnects the VMs in a virtual cluster logically across numerous physical networks. Physical or virtual machines can be used as virtual cluster nodes. On the same physical node, many VMs running various operating systems can be deployed. A virtual cluster's number of nodes can increase or shrink dynamically, similar to how the size of an overlay network varies in a peer-to-peer (P2P) network. As VM nodes are added, deleted, or migrated dynamically over time, the boundary of a virtual cluster can be changed.

Shared Storage: Shared storage is a type of high-capacity storage system that all users can access. Files are commonly used to communicate between workflow tasks, with each job producing one or more output files that are then used as input files by other tasks. These files are exchanged between processes using a shared storage system or some other method of file transfer. These storage systems must scale effectively to handle data from several workflow tasks executing in parallel on distinct nodes in order to achieve high performance. Workflows can typically involve a high-performance, parallel file system like Lustre, IBM GPFS (General Parallel File System), or Panasas when running on HPC machines. IaaS users in the cloud require a file system that can be accessed by all VMs at the same time. A shared disk file system or a distributed file system can accomplish this. In a shared-disk file system, all of the cluster's computational nodes share data storage that is often hosted remotely (de Oliveira et al., 2019).

Workflows in the cloud have the option of using a cloud storage service or deploying their own shared file system. To use a cloud storage service, the workflow management system would almost certainly have to change how it handles data. To use Amazon S3, for example, a workflow task must first collect input data from S3 to a local disk, executes, and then deliver output data from the local disk to S3. Making several copies in this manner can have a negative impact on workflow performance. Some other option is to configure a cloud-based file system that can be accessed by the workflow. In Amazon EC2, for example, an additional VM can be launched to host an NFS file system, and worker VMs can mount that file system as a local partition. In case more speed is required, multiple VMs can be launched to host a parallel file system such as PVFS (Parallel Virtual File Systems) or GlusterFS (Juve & Deelman, 2010).

It is not enough to have a collection of virtual computers to perform a workflow. Binding workflow tasks to resources for execution requires the use of additional software. The simplest

method to accomplish this is to employ a commercial resource management system such as Condor, PBS, or Oracle Grid Engine (Juve & Deelman, 2010). As a result, the virtual cluster resembles a traditional computer cluster. Such virtual cluster is made up of a master node that manages the other machines in the cluster and receives jobs from the workflow management system, as well as a group of worker nodes that perform tasks. Contextualization includes the deployment of the resource manager and the registration of worker nodes with the master node (Juve & Deelman, 2010; Keahey & Freeman, 2008).

The need of cloud customer is to complete an execution of a number of tasks at a low cost. As a result, the reliability and availability of the resources to be mapped must be confirmed. Aside from the fact that resources are limited, it can be difficult to provide the desired service in a timely manner. As a result, CSP must efficiently schedule its resources in order to provide the best possible service to its clients. Heterogeneity, dynamism, and uncertainty must all be controlled while scheduling resources. There are also other challenges to consider, such as ensuring that SLAs are not broken and maintaining QoS. Performance, reliability, availability, cost-effectiveness, and other factors were important to cloud users. Reduced energy consumption, finding a way to reduce rental costs while maintaining system performance, and so on are all things that providers are required to do.

Service Level Agreements (SLAs) are commonly used by cloud providers to provide end users with assurances about the quality of their services (Mastroeni et al., 2019). The obligations put on the cloud provider are indicated by a collection of service quality metrics and constraints to be met in those agreements that comprise a contract or a contract-like arrangement between the parties. The provider must ensure that SLA requirements are met by meeting the needs of customers.

Both efficient resource provisioning and scheduling are required to manage these resources. The efficient matchmaking of current resources to workloads is a major issue with Cloud Computing resource management. User workloads are monitored by resource management, which assigns tasks to available resources in accordance with Quality of Service (QoS) criteria. Workload Resource Manager (WRM) is a component that is responsible for this task. Resource Provisioning, Resource Scheduling, and Resource Monitoring are the three components of resource management.

After a user or a broker on behalf of a user submits his or her workloads to CSP via the Cloud Workload Management Portal, resources are allocated based on the information provided by the customer (Gill & Chana, 2016a). This procedure is called resource provisioning. From this point forward, it is the responsibility of the Resource Management System (RMS) to monitor and keep track of the workload submitted and the resources required until the workload is successfully executed without violating the SLA (Vijaya & Srinivasan, 2016). The classical resource management technique has been described as addressing fundamental issues such as service quality, economic approach, and resource utilization. Resource provisioning is the appropriate detection, selection, deployment, and management of resources to customers' tasks or workflows in a way that conforms to the QoS. SLA must be taken into consideration when allocating resources.

Only if the requested resources are there in the resource pool, does the resource provisionor supply them to the client workloads. If the requested resources are not available in the resource pool, the client must resubmit his or her task to the Workload Management System (WMS) with new QoS requirements. This time, however, the QoS criteria has to be changed. Resource allocation should be as close as possible to the present demand of clients' applications. Under-provisioning can leads to violation of SLA and then QoS. Over-provisioning also follows wastage of resources, energy and money.

Scheduling in grid and cluster environments is based on a best-effort approach, in which a user describes their computation (the number of resources and time needed) and assign responsibility for allocating resources and scheduling the computation to a batch scheduler. Requests for resources are automatically queued and processed in order as resources become available, according to the scheduler's policies. Consequently, jobs frequently encounter long, unpredictable queue waits, particularly those requiring a large amount of resources or having long runtimes. The allocation of resources and the assignment of work to those resources are closely tied and beyond the user's control. This is undesirable for workflows because the overheads of scheduling jobs on these platforms are often significant, and for a workflow with multiple tasks, the penalty is paid several times, lowering performance. The process is reversed in clouds. Rather than assigning allocation to the resource manager, the user directly provisions the resources needed and schedules their computations using a user-controlled scheduler. This provisioning mechanism is suitable for workflows and other loosely connected applications

because it allows the program to assign a resource once and utilize it to execute multiple tasks, reducing total scheduling overhead and improving workflow speed significantly (Juve & Deelman, 2010, 2008). The resource scheduler is in charge of user workloads after resource provisioning is completed successfully. Static provisioning, dynamic provisioning, and user self-provisioning are the three types of resource provisioning available. Resource provisioning is out of the context of this paper.

The cloud scheduler manages VM requests, gathers resource information from all computing nodes (such as available CPU nodes, available storage size, price, and so on), maps each VM request to the appropriate computing node, and allocates the resources required for VM requests based on the QoS requirements. The Resource Scheduler is in charge of mapping submitted cloud user workloads to available resources based on the information provided. That is, identifying the proper cloud resource (or combination of resources) and efficiently mapping workloads to it. Resource Allocation and Resource Mapping are the two functions of the Resource Scheduler. Resource Allocation is the process of allocating appropriate cloud resources to user workloads on a timely basis. This guarantees that resources are used efficiently. While Resource Mapping maps user-provided workloads to appropriate resources based on QoS settings submitted by cloud customers in terms of service level agreements (SLA) (Gill & Chana, 2016b). This mapping should be done in such a way that it increases throughput while maintaining compute load balance, avoiding underloading and overloading. Effective workload scheduling is crucial in a cloud environment to ensure effective task execution with the smallest number of resources possible in the shortest amount of time and with the best potential budget and resource efficiency. Effective cloud resource mapping is necessary for effective cloud resource scheduling (Gill & Chana, 2016a). Finally, Dispatcher dispatches workloads for execution, but only if the workloads are capable of meeting the stated QoS requirements.

Resource Monitor is another component of Resource Management, which controls resource utilization. Throughout workload execution, a monitor agent checks if the workload is being executed with an expected number of resources. If the Provided Resource (PR) is less than the Required Resource (RR), then the monitor agent asks for more resources to be provided (Gill & Chana, 2016b; Guruprasad & H, 2014). By rescheduling, resources in the resource pool is provided for the workload to be successfully completed. On successfully completion of workload execution, resources will released back to the resource pool.

2.8 Task Scheduling in Cloud Computing

The volume of data and tasks to be processed has increased considerably as the use of cloud computing services has grown, requiring numerous system resources and frequently resulting in severe resource waste. As a result, we should better schedule these data and operations. A task is a unit of execution or work that consists of a series of operations and requires specific computer resources to complete. Resource scheduling can be divided into two parts: task scheduling and VM scheduling (W. Lin et al., 2016). Task scheduling is the focus of this research paper. Task scheduling in cloud is the approach of determining the best execution sequence for jobs or activities. It is the assignment of tasks to a collection of resources that can be shared across various administrative domains by a task scheduling component. The scheduler's primary goal is to determine which task should execute first among a group of tasks, and on which resources, in order to maximize resource utilization and reduce execution time. Scheduling algorithms must be efficient for improved system performance and overall load balancing. Interruption management, fault tolerance, and minimizing total execution time should be considered.

Scheduling algorithms have a direct impact on the potential of client businesses and frequently allow for the efficient use of resources that perform operations in cloud computing environments. To achieve high performance, lower energy consumption, and lower costs, cloud data centers must manage physical and virtual infrastructure. The current resource scheduling in cloud data centers uses traditional resource allocation methodologies, making these goals impossible to implement. Because datacenters are geographically dispersed, issues such as long communication delays and resource underutilization are difficult to handle clearly and efficiently. Response time, datacenter processing time, communication latency, and throughput are all included in performance evaluation metrics. Many resources do not participate in executing requests, resulting in an imbalance in the cloud system.

Cloud data centers face scheduling challenges such as dynamic flexibility in overall performance in the distribution and migration of VMs and PMs, overall balance (CPU, storage, and network) and other resource considerations, rather than a single factor; resolution of discrepancies in system performance specifications; and energy efficiency and cost-effectiveness. To meet user expectations, the scheduler requires a specific technique for

mapping jobs to appropriate cloud resources. However, the majority of these scheduling techniques are static scheduling algorithms. Given the status of cloud resources, they give a reasonable schedule that does not take into account changes in resource availability. Dynamic scheduling, on the other hand, takes into account the current condition of the system. It has a flexible architecture and is capable of establishing productive schedules, which effectively reduces work completion time and improves overall system performance. When dealing with critical problems, dynamic scheduling considers multidimensional resources (CPUs, storage, and networking), load balancing, energy consumption, utilization, and other factors, rather than just static, predetermined parameters. The problem is its run-time overhead.

However, due to a variety of factors, scheduling and executing workflows in the cloud remains an unresolved but critical issue. To begin with, the cloud's pricing model must be set in order to fit under the budget guided by scientists. Cloud providers, on the other hand, enable for the acquisition of resources on demand, based on a pay-per-use pricing model, e.g., customers pay according to the used-time quantum (e.g., minutes, hours, days, etc.). Second, because clouds are changing environments, they may be sensitive to performance fluctuations during the workflow's execution, requiring adaptive/dynamic scheduling solutions. The ability to scale resources elastically is a major feature of clouds. The provider allocates and reallocates resources as needed to deliver this capability, and users are not aware of these changes unless they are administrators.

For example, if scientists use Amazon AWS spot instances, virtual machines can be unallocated or transferred as the provider need more CPU capacity in a certain data-center (de Oliveira et al., 2019). In the case of Amazon, demand varies throughout the year, which means that they may require more resources to manage the Thanksgiving and Christmas rush than they do the rest of the year. These movements and instability might have an adverse effect on the timing of workflow activations. Third, virtual machine failures are no longer an exception in the cloud, but rather a feature of the system. Although the cloud provider makes every effort to ensure the environment's reliability, workflow scheduling must account for possible failures when scheduling activations to run on several distributed virtual machines (de Oliveira et al., 2019).

2.9 Classification of Scheduling Algorithm in Cloud

Many researchers categorize different scheduling algorithm into different categories based on the focus of study and goal or destination which they want to reach (Vijaya & Srinivasan, 2016). As shown in figure 2.6, based on the layer of the service stacks, scheduling approaches can be divided into High-level Taxonomy and Low-level Taxonomy. High-level taxonomy mainly focuses on architecture of the Cloud service stack: IaaS, PaaS, and SaaS. Scheduling in application (software) layer, scheduling in deployment (infrastructure) layer and scheduling in virtualization (platform) layer (Zhan et al., 2015). Low-level taxonomy mainly focuses on scheduling objectives and further subdivides high-level taxonomy into more divisions. In this case, scheduling in application layer subdivided into scheduling for user QoS, provider efficiency and for negotiation. Scheduling in deployment layer subdivided into scheduling for service placement, partner federation and data routing. While, Scheduling in virtualization layer categorized into: scheduling for load balance, energy conservation and scheduling for cost effectiveness.

Other researchers also classify scheduling algorithm based on the information required for scheduling. Based on this information and the time of scheduling decision is made, they divide scheduling algorithm into two. Dynamic and Static Algorithm (Chenni Kumaran & Aramudhan, 2018). The static scheduling algorithm has less overhead than dynamic scheduling algorithm whereas the performance of dynamic scheduling algorithm is much better as compared to the static scheduling algorithm. In Static Scheduling (pre-run-time), every necessary information like precedence constraints, deadlines, maximum execution time and mutual exclusion constraints, for an execution of job is available prior to task execution starts, i.e. decisions are made at compile time (off-line).

Therefore, every information to enable the dispatcher to decide which job is going to be scheduled next at every point of a discrete time-base is available in the dispatching table. It is easy to implement and less run-time overhead (Cottet et al., 2002). Dynamic Scheduling on the other hand, contains no information about the task component before execution of the task. It makes its decision at run-time. Since it assigns resources at run time, opposing to static, which assigns resources before execution starts, it is more flexible but more run-time overhead. By selecting one task at a time from the ready queue, it automatically generates scheduling. There

is many scheduling algorithms used in distributed environment at this point. After performing appropriate analysis, many of those algorithms are suitable for cloud systems. Task scheduling helps to get better throughput as well as enhanced performance.

Based on the type of data: task scheduling and workflow scheduling. A task is a single isolated process, whereas a workflow is a complicated end goal that is accomplished by completing the sub-tasks of its portion. On the basis of method used: deterministic and stochastic algorithm (Vijaya & Srinivasan, 2016). In deterministic algorithm, computation is done based on certain mathematical model and cannot be modified then. In stochastic algorithm but, there is no predefined rule and it uses random search techniques. Scheduling can be implemented different techniques: scheduling VM and scheduling host. In the scheduling of VM, by depending on the incoming task the VM is assigned to the user requests, while in the host scheduling the number of VM is scheduled by taking the need of the host into a consideration (Ul Islam & Rana, 2017). Resource scheduling that takes place at VM-level is the mechanism that maps tasks to the allocated VMs by the help of job scheduler and this is called Task scheduling. While scheduling that takes place at the host-level is the allocation of VMs into physical machines with the help of VM scheduler. It is called VM Scheduling (A & K, 2016). There are also many other classifications of scheduling algorithms based on different schemes. Heuristics and Meta-heuristics types of algorithms are discussed below.

2.9.1 Heuristic Algorithms

To provide solutions to real-world scheduling problems, we must relax restrictions on the parallel program and the target machine representations. Recent research in this area has emphasized heuristic approaches. A heuristic produces an answer in less than exponential time but does not guarantee an optimal solution. Intuition usually helps us come up with heuristics that use special parameters affecting the system indirectly. A heuristic is said to be better than another heuristic if solutions approach optimality more often or if a near-optimal solution is obtained in less time. The effectiveness of these scheduling heuristics depends on several parameters of the parallel program and the target machine. A heuristic that can optimally schedule a particular task graph on a certain target machine may not produce optimal schedules for other task graphs on other machines. As a result, several heuristics have been proposed, each

of which may work under different circumstances (de Oliveira et al., 2019). There are three types of heuristics.

List Based Heuristic - In general, list scheduling is a class of scheduling heuristics in which tasks are assigned with priorities and placed in a list ordered in decreasing magnitude of priority. Whenever tasks contend for processing, the selection of tasks to be immediately processed is done on the basis of higher priority (Arabnejad & Barbosa, 2014). Priority tasks being assigned to resources first produce the most efficient schedules, without compromising the makespan and with a complexity that is generally quadratic in relation to the number of tasks. The differences among various list heuristics mainly lie in how the priority is defined and when a task is considered ready for assignment (Dong, 2009). An important issue in DAG scheduling is how to rank (or weigh) the nodes and edges (when communication delay is considered). The rank of a node is used as its priority in the scheduling. Once the nodes and edges are ranked, task-to-resource assignment can be found by considering the following two problems to minimize the makespan: how to parallelize those tasks having no precedence orders in the graph and how to make the time cost along with the critical path in the DAG as small as possible.

HEFT selects the task with the highest upward rank at each step (an upward rank is defined as the maximum distance from the current node to the exiting node, including the computational cost and communication cost). The selected task is then assigned to the processor which minimizes its earliest finish time with an insertion-based approach that considers the possible insertion of a task in an earliest idle time slot between two already-scheduled tasks on the same resource. HEFT has a complexity of $O(v^2.p)$, where v is the number of tasks and p is the number of processors. HEFT may be one of the listing algorithms, which aim to reduce the makespan of tasks in a DAG that is most frequently referred to. For example, it is tested in ASKALON and compared with a genetic algorithm and a myopic algorithm; the results show its effectiveness in the Grid scenarios, especially when the task graph is unbalanced. A critical issue in list heuristics for DAGs is how to compute a node's rank. In a heterogeneous environment, the execution time of the same task will differ on different resources and the communication cost via different network connections will differ as well. Thus, the rank of a particular node will also be different if it is assigned to different resources. The problem is how to choose the proper value to make the ordering decision. These values could include the mean value (like the original HEFT), the median value, the worst value, the best value and so on.

Clustering Heuristics - are mainly proposed for homogeneous systems to form clusters of tasks that are then assigned to processors. Clustering is another efficient way to reduce a communication delay in DAGs by grouping heavily communicating tasks to the same labeled clusters and then assigning tasks in a cluster to the same resource. Clustering tasks on only a few resources means low resource utilization. For heterogeneous systems, CHP algorithms (Vijaya & Srinivasan, 2016) and Triplet (Gill & Chana, 2016b) were proposed, but they have limitations in higher heterogeneity systems. In general, clustering algorithms have two phases: the task clustering phase that partitions the original task graph into clusters, and a post-clustering phase which can refine the clusters produced in the previous phase and get the final task-to-resource map.

Usually, each node in a task graph is an independent cluster at the beginning. In each clustering iteration, previous clusters are refined by merging some clusters. Generally, clustering algorithms map tasks in a given DAG to an unlimited number of resources. In practice, an additional cluster-merging step is needed after clusters are generated, so that the number of clusters generated can be equal to the number of processors. The problem of obtaining an optimal clustering of a general task graph is NP-Complete (Derouiche et al., 2019), so heuristics are designed to deal with this problem.

Duplication Based Heuristics - An alternative way to shorten the makespan is to duplicate tasks on different resources, but they have two disadvantages: a higher time complexity, i.e., cubic, in relation to the number of tasks, and the duplication of the execution of tasks, which results in more processor power used. This is an important characteristic not only because of the associated energy cost but also because, in a shared resource, fewer processors are available to run other concurrent applications. The main idea behind duplication based scheduling is utilizing resource idle time to duplicate predecessor tasks. This may avoid the transfer of results from a predecessor to a successor, thus reducing the communication cost. Duplication based algorithms differ according to the task selection strategies for duplication.

Originally, algorithms in this group were usually applied for an unbounded number of identical processors such as distributed memory multiprocessor systems. According to the basic idea of task clustering, cluster heuristics need not consider heterogeneity of the resources in the clustering phase. However, in the following cluster merging and resource assigning phases,

heterogeneity will definitely affect the final performance. Metaheuristic methods have been applied to solve task assignment problems in order to reduce makespan and response time. The methods have proven to find an optimum mapping of tasks to resources, which reduce the cost of computation, improve quality of service, and increase utilization of computing resources.

2.9.2 Metaheuristic Algorithms

Metaheuristics, in their original definition, are solution methods that orchestrate an interaction between local improvement procedures and higher-level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space. Over time, these methods have also come to include any procedures that employ strategies for overcoming the trap of local optimality in complex solution spaces, especially those procedures that utilize one or more neighborhood structures as a means of defining admissible moves to transition from one solution to another, or to build or destroy solutions in constructive and destructive processes.

ACO, PSO, GA, and their variants are the mostly commonly used nature inspired population based algorithms in the cloud. PSO outperforms GA and ACO in most situations and has faster execution time. PSO is simple to implement as compared to GA and ACO respectively. Workflow scheduling problems have been widely studied using PSO with the aim of reducing communication cost and makespan. Scheduling of Independent tasks has also been studied in cloud using PSO and it has proved to ensure minimal makespan. Improved and hybrid versions of PSO were also proposed for scheduling of tasks in the cloud, and they obtained better solution than those of ACO and GA. Symbiotic Organism Search (SOS) algorithm is a novel population-based metaheuristic algorithm for solving numerical optimization problems on a continuous real space. SOS mimics the symbiotic associations (mutualism, commensalism, and parasitism) among different species in an ecosystem.

2.10 Scientific Workflows

Workflows are coarse-grained parallel applications made up of a series of computing tasks that are logically linked by data- and control-flow dependencies. In a distributed computing context, workflow is used as an appealing framework (Juve & Deelman, 2010). Because workflow activities frequently analyze large amounts of data, we consider that each workflow activity may correspond to multiple executable tasks in order to investigate data parallelism and parameter sweep. Each task is the smallest processing unit, and it can run in parallel by consuming a

distinct portion of the input data (Teylo et al., 2017). In the context of this study, a task represents the atomic execution of an activity that takes different set of parameter values, a data partition or chunk. Workflows denote a wide range of application domains such as HPC (high performance computing) applications, interpretation applications, multi-tier Web applications, scientific computing, and large data processing applications. They are used to integrate multiple computational processes into a single coherent whole.

Generally, there are two types of workflows: Simple and Scientific workflows. Today different fields of sciences are using computer simulation for the purpose of different complex scientific experiments that works on a serious of data transformations. Such experiments consume and produce huge amount of data. We can refer such experiments *in silico* experiments (de Oliveira et al., 2019; Teylo et al., 2017). Scientific researchers in many areas like: earthquake science, computational fluid dynamics, bioinformatics, geophysics, phylogenetic analysis, high-energy physics, astronomy, weather monitoring, and gravitational wave physics are taking the advantage of Cloud Computing providing them high computing power, storage and bandwidth. Workflows may easily represent many various types of scientific analysis, and as a result, they are extensively used to model computations in many research disciplines. (Juve & Deelman, 2010; G. Kaur & Mathai, 2018; Marozzo et al., 2017). These experiments are mostly represented as a chained of scientific programs together to feed each other, i.e. output of one program can be input to another program.

For the proper management of execution of these complex experiments and structuring data analysis, workflow can be a comprehensive tool (Mimura Gonzalez et al., 2017; Teylo et al., 2017). For instances, a lot of scientific workflow applications include: Montage, Epigenomics, CyberShake, SIPHT (sRNA Identification Protocol using High throughput Technology) and LIGO (Laser Interferometer Gravitational Wave Observatory) (Juve et al., 2013). A number of existing large-scale scientific experiments based on simulations produce very massive output files. It requires enormous amounts of computational resources to execute the experiment. It is common to have multiple activities in the same workflow, each of which may communicate with another task therefore; it is a data-intensive application. Likewise, as a single task in such data-intensive workflows can take several hours or even days to complete, it can also be compute-intensive. These data- and compute-intensive workflows, if run sequentially, could

take several months to complete, which is error-prone and not acceptable in today's competitive science environment (Teylo et al., 2017).

2.11 Scientific Workflow Examples

2.11.1 Montage

Montage was created by the NASA/IPAC Infrared Science Archive as an open source framework that can be used to create custom sky mosaics using Flexible Image Transport System (FITS) input images (Juve et al., 2013). Montage is known for its computing application for the construction of custom astronomical image mosaics of the sky in the area of astronomy (de Oliveira et al., 2019). This data-intensive toolkit is used for modelling a workflow, which is the outcome of the National Virtual Observatory Project, which stitches tiles of sky photographs from different sky surveys into a photo-realistic single image. The geometry of the output image is determined from the input images during the creation of the final mosaic. The input images are then re-projected to have the same map coverage and rotation, the background emissions in the images are corrected to have a consistent amount, and the re-projected, corrected images are brought together to create the output mosaic.

The scale of the Montage workflow depends on the quantity of images used to create the required mosaic of the sky. See fig 2.8 below for the composition of small Montage Workflow. Each node corresponds to the execution of a scientific application. The configuration of the workflow adjusts to reflect an increase in the number of inputs, which leads to an increase in the number of computational jobs.

Montage has a number of tasks that can be executed in parallel, as they are parallel tasks and every task requires different CPU, disk, and memory requirements. Consequently, the framework in which the workflow is executing has to define which tasks will execute in which machines. In regards of data, a small instance of Montage workflow utilizes and creates many gigabytes of data, and Montage needs to allow all input images to be on the local disk and to write the output mosaic to the local disk. The larger part of its runtime, Montage Workflow spent in performing I/O on average. According to Gideon Juve et al. (Juve et al., 2013) shows low CPU utilization for several tasks in Montage Workflow because of Montage jobs spend much of their runtime on I/O operations. According to this experiment, only one job

(mBgModel) had the extensive average job runtime, which is the highest CPU utilization (99.89%) and low I/O activity.

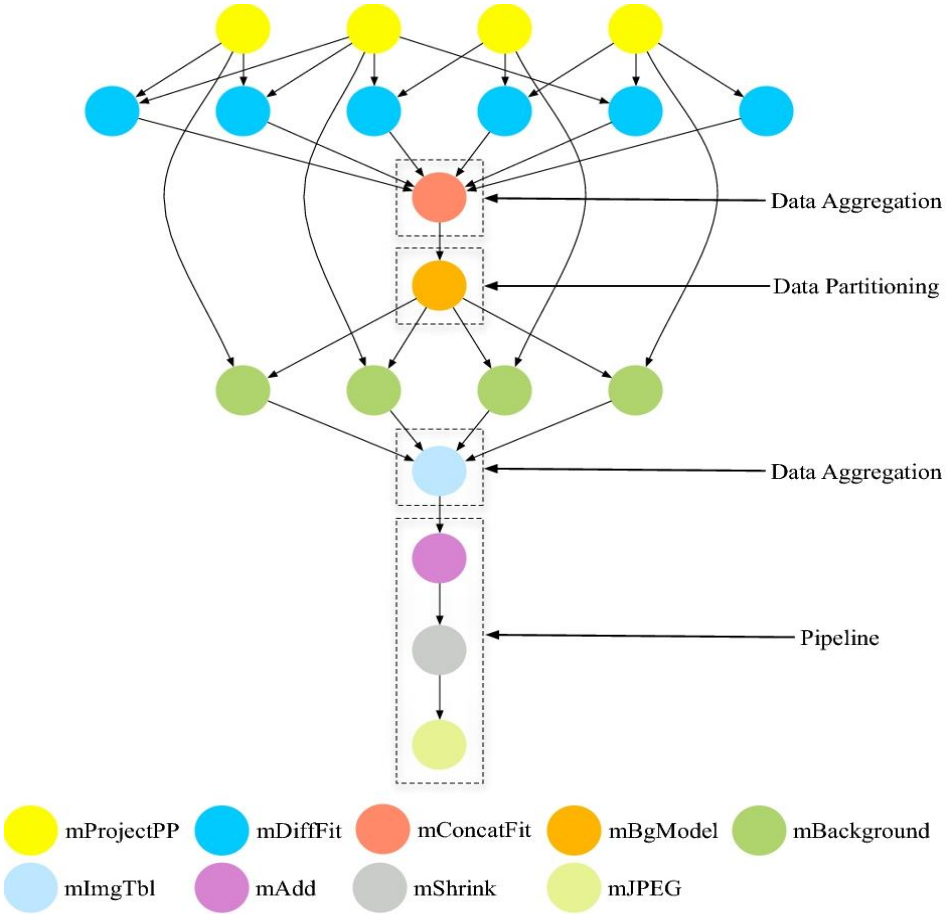


Figure 2. 2 The Montage workflow[1]

2.11.2 Epigenomics

The USC (University of Southern California) Epigenome Center is actively interested in mapping the epigenetic state of human cells on a genomic scale (Bharathi et al., 2008). The Epigenomics workflow is essentially a highly data processing pipelined application that operates on independent chunks of data in parallel with several pipelines. It uses the Pegasus Workflow Management System to make it easy the execution of different genome sequencing operations (Juve et al., 2013). DNA sequence data obtained from genetic analysis process for several lanes is an input to the Epigenomics Workflow. The Illumina-Solexa Genetic Analyzer system generate the DNA sequence data, which is broken down into many chunks that can be run in

[1] https://pegasus.isi.edu/workflow_gallery/

parallel. The data in each chunk is translated to a file format understood by the Maq DNA sequence mapping application. The remaining operations include the filtering out noisy and contaminated data from each of the chunks, mapping sequences into the correct location in a reference genome, converting the data into binary format for faster processing and low disk space utilization, generating a global map and then identifying the sequence density at each position in the genome. Finally, the results of every individual map processes are merged and the data is reformatted so it can be displayed on a GUI application.

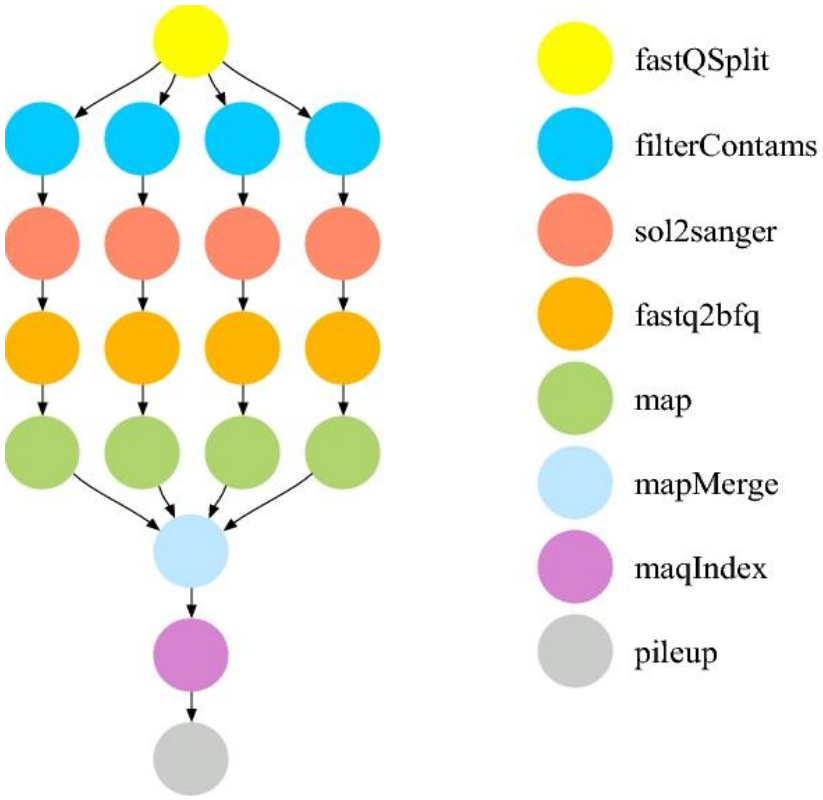


Figure 2. 3 The Epigenomics workflow[2]

Mapping activities that match sequences with the reference genome are the most computationally expensive, i.e. it is CPU-bound workflow. Mostly Epigenomics workflow is highest CPU utilization; some job has an average utilization of 153% (pileup). It is possible for a job to have a CPU consumption of more than 100% if several cores are used. Map tasks constitute the overwhelming majority of the runtime of the Epigenome workflow, i.e. approximately about 96%.

[2] https://pegasus.isi.edu/workflow_gallery/

2.11.3 LIGO Inspiral Analysis Workflow

The Laser Interferometer Gravitational Wave Observatory (LIGO) inspects data from the coalescing of compact binary schemes, such as binary neutron stars and black holes and tries to detect gravitational waves created by various events in the universe according to Einstein's theory of general relativity (Juve et al., 2013). It is very complicated and consisting of many sub-workflows. The real workflow profiled includes 25 sub-workflows of more than 2,000 overall tasks. Other workflows have up to 100 sub-workflows and 200,000 tasks. Time-frequency data from each case for any of the three LIGO detectors is broken down into smaller blocks for analysis. The workflow produces a subset of waveforms belonging to the parameter space and computes the output of the corresponding filter for each block. If a true Inspiral has been identified, a signal is produced, which can then be tested with triggers for other detectors. Several further accuracy tests can also be carried out.

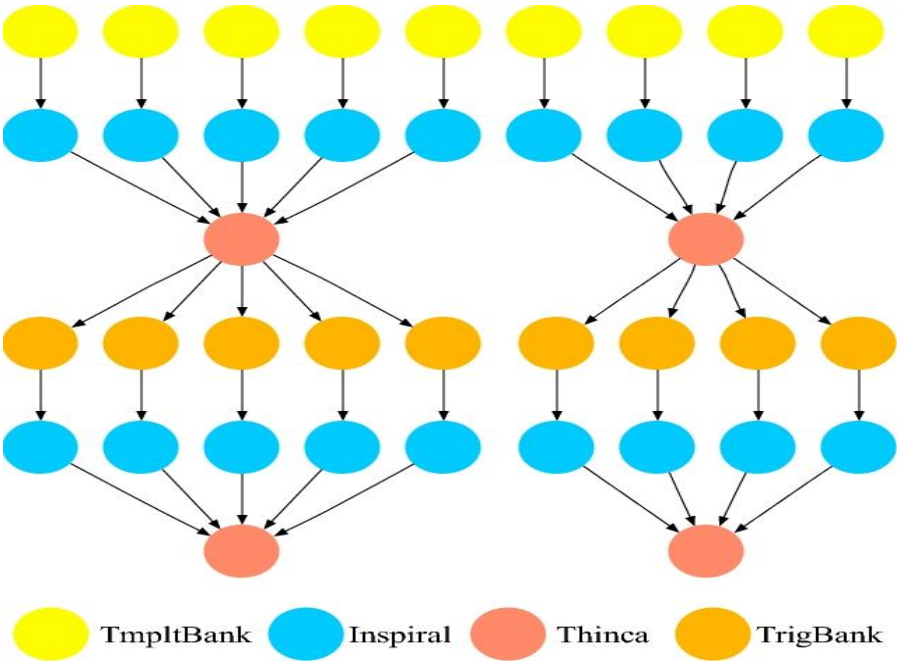


Figure 2. 4 LIGO Inspiral Analysis Workflow[3]

LIGO Inspiral jobs that run the matched trigger filter are the most computationally intensive workflow jobs; 358 instances of Inspiral function collectively consume approximately about

[3] https://pegasus.isi.edu/workflow_gallery/

92% of the overall runtime and approximately about 92% of the I/O (197 GB) and have reasonably high CPU efficiency (~90%). The 29 TmpltBank jobs are also computationally intensive (average runtime of 500 seconds and average CPU usage of ~99%) and read important input data (16 GB). A good point in LIGO Inspiral Analysis Workflow is it reads a large amount of data (213 GB), but writes very little (50 MB).

2.11.4 SIPHT

SIPHT is bioinformatics project at Harvard University and it is high throughput computing technology for workflow management used for an automation of kingdom-wide search for small untranslated RNAs (sRNAs) that control a variety of processes, such as bacterial secretion or virulence. It is used for forecasting, annotations of intergenic sRNA-encoding genes and searches for putative sRNA-encoding genes in all 1640 bacterial replicates in the National Center for Biotechnology Information (NCBI) database. Candidate sRNA-encoding loci are classified based on the occurrence of putative Rho-independent terminators downstream of conserved intergenic sequences, and each locus is annotated for several characteristics, including conservation in other species, association with one of several transcription factor binding sites and homology to any of more than 300 previously identified sRNAs and cis-regulatory RNA components. Using SIPHT, we looked for putative sRNA-encoding genes in all 932 bacterial replicons in the NCBI database. These searches yielded almost 60% of previously confirmed sRNAs, hundreds of previously annotated cis-coded regulatory RNA elements such as riboswitches, and over 45.000 novel candidate intergenic loci.

All SIPHT workflows have exactly the same composition and larger workflows can be made up of smaller, separate workflows. The only difference between two workflow instances is the number of Patser jobs that search sequences with position-specific score matrices that return corresponding positions; the number of Patser jobs depends on inputs specifying the transcription factor binding sites (TFBSs) [16]. Similarly, the SIPHT has a single job type (Blast) that accounts for most of its runtime (~74%). Likewise, similar to Epigenome, SIPHT is mainly a CPU-bound workflow. Most jobs have high CPU utilization and comparatively lower I/O utilization. Just only Patser_concate, which basically concatenates the performance of the Patser jobs, has low CPU consumption. Some of the workflow jobs (the Blast and Blast_QRNA) read the most input data (~800 MB each) and write the most output data (~565 MB each).

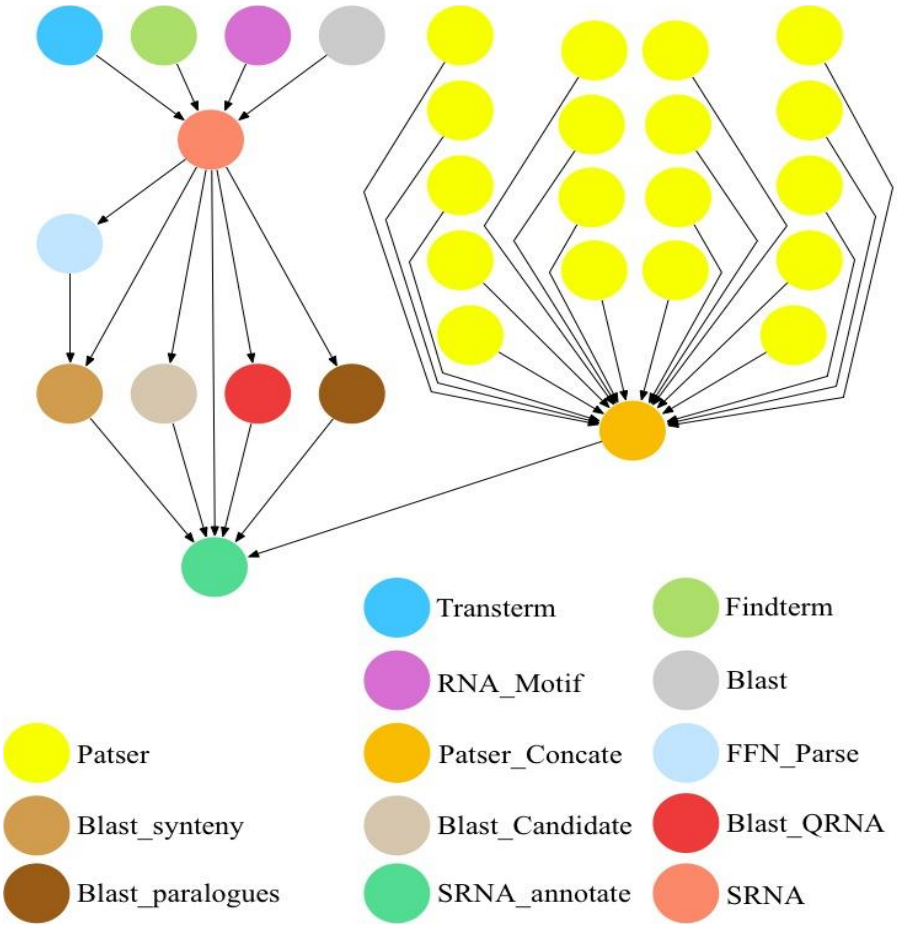


Figure 2. 5 The SIPHT Workflow[4]

2.11.5 CyberShake

The Cybershake workflow is an application of seismology that is used for identifying earthquake hazards by calculating Probabilistic Seismic Hazard curves for various geographic regions in Southern California Earthquake Center (SCEC) with the technique of Probabilistic Seismic Hazard Analysis (PSHA). It is used to classify all ruptures within a 200KM range. It often alters rupture into several rupture variants that vary in hypocenter positions and slip distributions. Synthetic seismographs are then measured for each rupture variance. The peak strength is extracted and applied to the original rupture probability for the development of probabilistic hazards for the region. Provided the area of interest, a finite difference simulation performed to produce Strain Green Tensors (SGTs). Synthetic seismograms are determined from the SGT data for each of the fractures that were expected. After that spectral acceleration

[4] https://pegasus.isi.edu/workflow_gallery/

and probabilistic hazard curves are generated. Preparation, simulation, post-processing, and analysis are all steps in the Cybershake methodology. There are two basic steps to the computing process. There is a simulation phase for earthquake wave propagation and a post-processing phase. CyberShake workflows resulting in a record of over 800,000 jobs have been executed using the Pegasus Workflow Management System (Pegasus-WMS) (Aziza & Krichen, 2020).

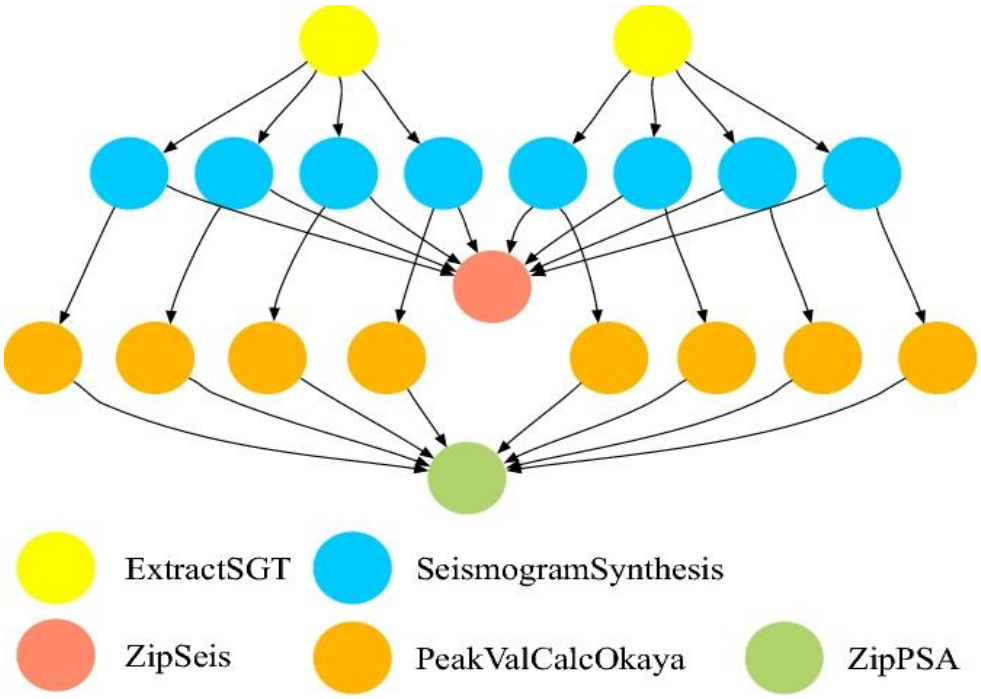


Figure 2. 6 The CyberShake Workflow

Of the CyberShake workflow computational jobs, seismogram synthesis jobs are the most computationally intensive one, i.e. it consumes vast majority of the runtime (>97%). Beside this, Seismogram-Synthesis reads, on average, 547 MB of data per job; however, the total size of all the input files for SeismogramSynthesis is known to be approximately 150-250 MB. It also needs very high memory requirements, i.e. on average, 817 MB of memory is needed, but the profiling data indicates that certain instances need up to 1870 MB of memory. This makes it to affect the scheduling of SeismogramSynthesis jobs because, for many of the nodes available on the grid, the amount of memory available per CPU core is only about 1GB. Thereby, if one SeismogramSynthesis job is scheduled for each core, the node be able to run out of memory. We must be careful if the application is moved to another cluster in the future to configure the

scheduler to never schedule more SeismogramSynthesis jobs on a node than the node can support.

2.12 Workflow Scheduling Problem

Workflow scheduling is a process of allocating concrete activations to computing resources (i.e., computing nodes) to be executed during workflow execution. The goal is to get an efficient SP (scheduling plan) that minimizes a cost function based on resource utilization, workflow execution cost, and execution time. Scheduling methods can be static, dynamic, or hybrid. Therefore, in many scenarios, an optimization algorithm is needed to achieve multiple objectives. The cloud provides several different computing resources and appears as appropriate infrastructure for executing data-intensive workflows. However, executing a workflow in the cloud is not a simple task. In cloud, first the SWfMS has to deploy the necessary resources, i.e., virtual machines provisioning is critical for workflow execution. Therefore, since workflow execution takes much time and cost, it is fundamental to achieve multi-objectives, i.e., reducing both execution time and financial cost when scheduling the activations to the virtual machines that are part of the virtual cluster (de Oliveira et al., 2019).

When the SWfMS can accurately estimate the execution burden of each activation, when the execution environment varies minimally during workflow execution, and when the SWfMS has enough knowledge about the processing and storage capabilities of the environment, it is efficient. Heuristic-based and guided random search-based processor selection methods are used in static activation scheduling algorithms. The heuristic-based method organizes tasks based on a predefined rule, whereas the random search-based method schedules tasks at random. Task-based and workflow-based static activation scheduling methods are two types of static activation scheduling algorithms. The task-based method assigns activations to computing nodes directly, whereas the workflow-based method assigns a fragment to computing nodes (de Oliveira et al., 2019). During workflow execution, on the other hand, dynamic scheduling generates SPs that distribute and allot executable activations to computing nodes. This type of scheduling is useful for workflows where estimating the workload of activations is difficult, or for situations where computer capabilities change greatly during execution (i.e., clouds).

When executing a data-intensive workflow in the cloud, each task must be scheduled on a specific VM. Scheduling consists of deciding when and where to execute all tasks. To schedule

tasks across distributed computing resources, even in basic contexts, is an NP-complete issue. Clouds, on the other hand, have some properties that make the scheduling procedure a little bit more difficult (Teylo et al., 2017). In dynamic execution contexts, dynamic scheduling can provide load balancing, however it takes time to produce SPs during dynamic execution. In a publish/subscribe architecture, the scheduling algorithms can be based on sorted queue approaches with diverse strategies such as First-In-First-Out (FIFO), adaptive, and so on. Both static and dynamic scheduling have benefits, and they can be used as a hybrid scheduling strategy to obtain greater results than either one alone. For example, a SWfMS might use static scheduling to arrange portion of a workflow's activities, such as those activations for which there is enough information, and dynamic scheduling to schedule the rest during execution (de Oliveira et al., 2019).

Scientific Workflow has emerged as the main solution for management of such scientific processes. Workflow is a set of interdependent tasks, which are the smallest unit of processing. These scientific applications represented as a directed acyclic graph (DAG) in which nodes, vertexes represent computations (tasks) and the edges, or arcs describe data and control dependencies between them. Several existing large-scale scientific experiments based on simulations produce very large files and datasets and require huge amounts of computing resources to execute (de Oliveira et al., 2019). For this matter, this data intensive scientific application requires flexible computing resources depending on the requirements and budget of users.

Therefore, scientific workflow scheduling is mapping tasks of the workflow onto virtual machines with data dependency constraints aiming at lesser execution time and maximizing resource utilization. However, it is a challenging problem to effectively schedule tasks of scientific workflow in Cloud environment (G. Kaur & Mathai, 2018). Since a single unit of workflow may comprises of a set of tasks each of which may communicate with data transfer, there must be constraints to be satisfied. This data dependency constraint among tasks of workflow makes it tough to achieve maximum utilization and minimum makespan.

2.13 Scientific Workflow Management System

In a number of computing scenarios, a SWfMS is a powerful and sophisticated tool for planning, executing, monitoring scientific workflows and managing data (de Oliveira et al., 2019; Juve & Deelman, 2010). It adapts workflows to the target execution environment, organizes task

execution on distributed resources, optimizes workflow performance to reduce time to solution and speed up the production of scientific results, and ensures execution dependability. As a result, scientists do not have to deal with a high number of failures, and they keep track of data so that it can be found and used simply throughout and after workflow execution. Building a virtual cluster using cloud resources and scheduling workflow tasks to that cluster is one way to run workflows on the cloud.

Computing provisioning refers to the provisioning of computing nodes to SWfMSs, whereas storage provisioning refers to the provisioning of storage resources for data caching and persistence. During workflow execution, the data storage module often uses database management systems and file systems to manage the data. Taverna, for example, stores intermediate and output data in a database. Some SWfMSs employ a shared-disk file system, while others use distributed or local file systems on each compute node. In general, the provisioning module's computing nodes and storage resources are used by file systems and database management systems. SWfMSs can additionally store certain data in a computer node's disk cache to speed up data access during workflow execution (de Oliveira et al., 2019).

SWfMS	Spec.	UI	Parallelism	Scheduling
Pegasus	DAG	GUI/Textual	Data and Independent	Static and Dynamic
Swift/T	DCG	Textual	Activity	Dynamic
Kepler	DCG	GUI	Activity	Static and Dynamic
Taverna	DAG	GUI	Data and Activity	Dynamic
Chiron	DCG	Textual	Data, Activity, and Hybrid	Static and Dynamic
SciCumulus	DCG	Textual	Data, Activity, and Hybrid	Static and Dynamic
Galaxy	DCG	GUI	Independent	Dynamic
Triana	DCG	GUI	Data and Activity	Dynamic
Askalon	DCG	GUI	Activity	Dynamic and Hybrid
WPg	DAG	GUI	Data, Activity, and Hybrid	Static and Dynamic

Table 1 Comparison of SWfMSs

2.14 Related Works

The majority of cloud services remain unused if resource provisioning is just not done properly. The service provider uses Resource Allocation Policy, which is a mix of all the activities done to allocate scarce resources, to meet user requests and application needs. For this reason, a cloud

service provider must know how much and what kind of resources are required to complete a task. A service provider should be aware of all available resources as well as the state of each resource before provisioning of resources if happen. The user, on the other hand, should be aware of all the application's requirements. This is useful for avoiding scenarios like scarcity, fragmentation, and resource contention, as well as under provisioning and over provisioning. These issues will arise if multiple tasks attempt to use the same resource at the same time, a limited set of resources is accessible in comparison to user requirements, or a task is assigned to extra resources rather than an exact one. Because users' resource requests are unpredictable for service providers, proper resource management is required, and users seek services that take less time and money to perform.

H. Topcuoglu et al. (Topcuoglu et al., 2002) proposed two heuristic based list scheduling algorithms; Heterogeneous Earliest-Finish-Time (HEFT) and Critical-Path-on-a-Processor (CPOP) which generate a schedule in two main phases. Prioritizing the task and then processors are selected and allocated to the task. HEFT has achieved good performance when applied to workflow scheduling problems. However, HEFT shows poor flexibility to large-scale and complex task graphs. It does not consider the CPU load at runtime and it can fail easily to find optimal solution. Static heuristics, such as the min-min and Heterogeneous-Earliest-Finish-Time (HEFT) algorithms, schedule the entire task set using an anticipated cost. They could therefore find very good schedules via a thorough examination of task and resource features, even though they do not search the whole solution space or guarantee a "good" worst case. In traditional systems, both approximation and static heuristics are based on the premise that the resources are allocated such that the performance they provide remains constant during the execution of the entire task set. The HEFT algorithm does not explicitly account for data transport overheads. The main advantage of heuristics is that they provide solution in short period of time.

In (Toan et al., 2015) scheduling algorithm for multi-workflow Particle Swarm Optimization (PSO) is proposed to use multi-layer tools in Fog computing to minimize costs within specified fog timeframes. Different papers like (B et al., 2019) proposed different strategies of scientific workflow scheduling for different computational platform like grid and fog computing which directly cannot applied to cloud. Many cloud scheduling techniques, however, derived from these algorithms. However, there is additional research (Duan et al., 2014; Duan & Prodan,

2015) that focuses on the scheduling of ensembles with multiple constraints, which is different from our execution model, where we schedule workflows according to their priorities of tasks.

Workflows are widely used to model computations in many research disciplines because they can easily represent many different types of scientific analysis. Algorithms for scheduling single workflows onto generic execution units have been extensively investigated, and there are various works on the subject. Including (Teylo et al., 2017), (Nasonov et al., 2017), (Arabnejad & Barbosa, 2014), (Luiz Fernando Bittencourt et al., 2010), (Topcuoglu et al., 2002), (Durillo et al., 2015) and many more algorithms. Even though these techniques produce good results for single workflows, but they are not directly applicable in the context of IaaS clouds, where compute resources can be provisioned and de-provisioned on demand. On top of all that, they do not even take into account data movement between computational nodes, which is one of the most important that should be treated seriously. Data movement minimization is the focus of this research paper.

The problem of process scheduling with a single constraint or multiple objectives, such as cost, energy, deadline or storage, has also been addressed in many works. To tackle scheduling problem of scientific workflows in Cloud Computing environment many researchers have implemented a number of research works over the last decade. Among them data- and compute-intensive experimentation from different scientific fields are getting huge attention. Those research works have been focus on several computational requirements like data transfer strategies (Sukhoroslov, 2019) and management, total execution times (B et al., 2019; V. Singh et al., 2019) and parallelism level (Cieza et al., 2019). The majority of these researches prioritizing and aiming at reducing execution times of scientific workflow scheduling (Bryk et al., 2016).

To improve the execution time under reasonable costs, Singh et al. (R. Singh & Singh, 2013) suggest a score-based deadline constrained approach in which resources' capabilities are represented by a score. It sends a list of workflow tasks to the data center and retrieves the available virtual resources. The final scores of VMs are obtained from the component minimum sub-scores and opposite to our work, the lowest score VM is selected that meets the task's threshold, which is depending on the task's length. Workflow failure rates are reduced. However, tasks execution order is not addressed. To schedule workflow applications within deadline

constraints and at a reasonable cost, a new scheduling algorithm was proposed in (Chopra & Singh, 2013). To do this, they divide the workflow into stages with each level consisting of discrete tasks that are executed simultaneously. While the heaviest tasks require more processing is moved onto a private cloud, the lightest tasks can be transferred to public clouds where they can be executed on cheaper VMs with lower processing power. Comparing the suggested algorithm to the level-based technique and the min-min strategy, it is found to be less expensive. In this algorithm, the goal was to select the cheapest VM. This paper didn't consider time spent during the provisioning of these cheapest VM from the cloud.

In (Sukhoroslov, 2019) several data transfer techniques in the scientific workflow are compared and experimentally tested with their data transfer approach. Prefetch and Queue techniques have been designed for overlapping communications and computations, prioritizing data transactions and reducing network contention. It greatly increased the makespan. However, this work implements data transfer techniques aside from the job scheduling algorithm, which has never taken into account workflow scheduling. Wang et al. (Wang et al., 2014) suggests an algorithm that aims to minimize the transfer of data contained in several data centers. This paper describes the static and dynamic data generated during workflow execution, but the data transfer problem is not resolved.

The authors of (Bryk et al., 2016) suggest a dynamic scheduling algorithm for workflow ensembles, a paradigm that recognizes the nature of data storage and optimizes data transfer on the IaaS cloud platform. This paper notes that the transfer of data cannot be ignored and takes a significant amount of overall workflow execution time. The algorithm minimizes data transmission by prioritizing tasks that recognize the locality of the data. Tasks using same data file as input are mapped to the same virtual machine. Therefore, it could avoid unnecessary data transfer among virtual machines. Several approaches in the literature have been proposed on data placement for better management of data to minimize the overall execution time of workflow and total data movement (B. Lin et al., 2019), (Lizhen et al., 2015). Nevertheless, it all predominantly targeted to minimizing the data movement but not execution time of the all data placement process. In (Derouiche et al., 2019) a decentralized approach for data placement of both original and intermediate datasets is proposed for reducing the execution time of the data placement algorithm.

Durillo et al. (Durillo et al., 2015) introduced a new version of the Heterogeneous Earliest Finish Time (HEFT) workflow scheduling heuristic for dealing with numerous conflicting goals and approximate Pareto frontier optimal schedules. Performance and financial cost were used to evaluate this new version of HEFT. The authors used synthetic and real-world applications DISC and federated clouds environments. Arabnejad et al. (Arabnejad et al., 2015) present the Deadline–Budget Constrained Scheduling (DBCS) heuristic scheduling algorithm, which includes two significant constraints for workflow scheduling: time and cost, similar to Durillo et al. (Durillo et al., 2015). When utilizing DBCS, users must provide schedule and budget constraints, and DBCS will search for a feasible solution that meets both of these requirements. Durillo et al. (Durillo et al., 2013) present a bi-objective optimization strategy that takes both makespan and energy into account. For activity executions, the authors employed a Pareto-based workflow scheduling algorithm called MOHEFT, which used energy consumption and performance models.

In (Luiz Fernando Bittencourt et al., 2010) another variant of HEFT algorithm, which is DAG scheduling heuristics for heterogeneous resources, is proposed to enables HEFT to make more informed scheduling decisions with the help of lookahead in the schedule. It could minimize, in some cases, the makespan up to 20% on average. Another list-based heuristic (Arabnejad & Barbosa, 2014) with the same time complexity as the state-of-the-art algorithm improved the makespan and efficiency by using a look-ahead feature. Predict Earliest Finish Time (PEFT) has the same time complexity, $O(v^2.p)$, as HEFT and is based on computation of an optimistic cost table (OCT) for prioritizing tasks and assigning processors. In (Z. Liu et al., 2014) by merging the list scheduling (HEFT) with the task duplication scheduling scheme, a new heuristic, an earliest finish time duplication (EFTD) is proposed. They used fuzzy method for clustering of cloud resources to minimize the scheduling time for selecting processing unit. Whenever there is an idle time on the processor, EFTD algorithm tries to map the immediate parent nodes of the current selected node for the purpose of minimizing the waiting time on the processor.

Many literatures (Cieza et al., 2019; Teylo et al., 2017) mentioned important features for developing a workflow scheduler: (i) low scheduling overhead, (ii) aware of computing resources heterogeneity, and (iii) directly addressing the data file assignment problem. Teylo et al. (Teylo et al., 2017) suggested an evolutionary metaheuristic TaSDAP approach that treats

both tasks and data as vertices in workflow graphs. The authors suggest a new workflow format in which the DAG's nodes represent jobs or data files. For solving it, the authors proposed HEA-TaSDAP, a hybrid evolutionary algorithm. This algorithm tackles both the assignment of data files and the scheduling of assignments in accordance with local search strategies. It considers the variety and heterogeneity of VMs and also data distribution and data constraints altogether. HEA-TaSDAP could schedule instance of any size, but execution of larger tasks (instances larger than 500 tasks) may need several days to finish. Hence, not meeting scheduling overhead of above-mentioned features.

In work (Cieza et al., 2019), PHEA-TaSDAP is proposed to enhance the scheduling decision time with the help of enormous parallelism provided by GPU-based local search procedures. Optimizing execution time of HEA-TaSDAP by taking advantage of massive parallelism with the help of four GPU-based local search procedures, it could minimize the scheduling decision time by 98.64% with quality of the results. In (Arabnejad et al., 2015) a low complexity list scheduling algorithm proposed. It is scheduling algorithm for heterogeneous computing systems with complexity O(e) (p + log v), which produces effective results for DAG-based applications. By exploiting the advantage of Machine Learning, their capacity to find more efficient solutions than heuristic approaches, many approaches have been proposed to tackle task scheduling problems. According to (Yang, 2013), Y. Yang et al. presented V-heuristics for job allocation, such as V-MCT, which assigns every job in an arbitrary order of minimum completion time of the virtualized resource. In (Abrishami et al., 2013) IC-PCP (IaaS Cloud Partial Critical Paths) is proposed, which is based on the critical paths of the workflow, this algorithm creates subgroups of tasks to be executed. Scientists set a deadline, and VMs are assigned to each subset based on their financial cost and execution capabilities. There's a preference in the algorithm for selecting VMs that are already in use. The technique, however, deploys the cheapest machine that can execute the subset while still meeting the deadline if this is not achievable. Until there are no more tasks to schedule, the search for critical paths and the entire scheduling process is repeated. However, despite the fact that splitting tasks into subsets may lower data transfer costs, this approach does not study the task execution order.

This proposed work focuses on tackling both data file transfer problem that is enormously time-consuming task in workflow execution and scheduling of tasks within workflow. A score-based heterogeneous VMs scheduling algorithm that allocates resources to workflow application tasks

based on score of VMs performance attributes like processor speed, processing cores, storage capacity (secondary storage), bandwidth strength and RAM (primary storage) size. According to their capacity, different resources have different performance based on their MIPS, RAM, Storage, Processing Entity, and Bandwidth. The Scoring System is a component that evaluates and ranks the provided virtual machines (VMs) based on the listed attributes. The score approach, which indicates the system's minimal performance, can be used to measure the capabilities of cloud hardware resources, as it determines the capability of virtual machines. We can schedule workflow applications on high-performance, reliable machines with score-based scheduling, resulting in fewer workflow application failures.

The algorithm allocates VM with high score, i.e. the VM with high processing capabilities (MIPS), high number of processing cores, maximum size of both primary and secondary storage size and high bandwidth strength to tasks with highest priority. Tasks in HEFT are prioritized by the value of the upward ranks, $rank_u$, which is calculated based on the mean computation and mean communication costs of tasks. In this proposed work, to find the final rank $rank_f$ value of the task, the computation time of the task, the communication time between the tasks and the influence of the priority of the subtask on the priority of the parent task is taken into account. The final rank is computed for each task based on data transfer cost, computation cost, $rank_u$ and $rank_d$ of the task. By identifying data-intensive task, it will be mapped onto high score VM which is the same as its parent task. So that an overhead of data transferring can be tackled with this approach, which can makes makespan of workflow lesser.

CHAPTER THREE

RESEARCH METHODOLOGY

3.1 Design of the Study

Research design is a blueprint of research methods and techniques adopted by a researcher. It is a strategy for answering research questions. The activities of research design begin with problem formulation and proceed to the suggesting solutions. The quantitative research design using simulation of the scientific scheduling in cloud on WorkflowSim is used in this study. The simulation is performed using different synthetic scientific workflow applications published by Pegasus[5] project.

3.2 Data Collections

For the purposes of this research, mostly we used secondary data source such as, articles, journals, research papers, dissertations, web resources and other topic related documentations and guide lines. For the purpose of simulating our work, we collected synthetic scientific workflow applications from Pegasus web page. In a wide range of applications, Pegasus is now being used. The workflow runs are listed in Pegasus workflow gallery by application. The application page contains all runs and a short synopsis of the workflow. These synthetic workflows are similar to those used in scientific applications in the real world. For our simulation, four synthetic scientific workflows are chosen based on their resource consumption characteristics: CyberShake, Inspiral (LIGO), Epigenomics and Montage workflow. To conduct the research, we established an experimental environment using Apache NetBeans (Java) and the WorkflowSim tool. We have documented the results obtained by using the WorkflowSim simulation environment. Finally, we analyzed the findings and evaluated the performance of the proposed system by measuring the algorithm's running time using some existing metrics and comparing it to the existing methods.

3.2.1 Literature Review

In the literature review portion, we reviewed different job scheduling techniques. Every algorithm has its own set of advantages and disadvantages. Several algorithms are developed in order to overcome some of the limitations of previously existing algorithms. We conducted a

[5] https://pegasus.isi.edu

thorough literature review to fine-tune the state-of-the-arts as well as the scope of knowledge in the scientific workflow scheduling trends. Articles or complete text research from the early 2000s to the current year (2021) were chosen based on their relevance to the research. The sources were from several search engines. We selected the most relevant electronic sources to software engineering research topics, such as Google Scholar, IEEE Explorer, ACM digital library, SpringerLink, ScienceDirect, Scopus, and various research online databases, to conduct the automatic search. These online libraries are the most appropriate for the field of cloud computing because they give appropriate and simple search engine methods for the spontaneous search of resources in their database based on the combination of search term/string.

We used different search techniques to gather data from the internet for our study. The key sources of data for this research include websites, literature reviews, and previous research documents in order to acquire appropriate user-friendly design metrics. To collect relevant studies, the author used the search process, which consists of different staged searching procedures. There are different phases we used for online search procedures: manual search, automatic search, filtering, and data collecting. The search string is a combination of keywords, truncation symbols, and boolean operators. We used a search string, which is made up of the following strings:

(Cloud computing OR Cloud OR Scientific workflow OR Workflow OR Data-intensive) and (task OR workflow OR job OR ensemble OR heuristic) and (scheduling OR scheduler OR dispatch OR allocation).

3.2.2 Refining the Search Result

3.2.2.1 Inclusion Criteria

- The study should be in the field of cloud computing.
- The topic should be related to resource management, resource provisioning and/or scheduling techniques in cloud computing.
- The proposed solution given by each study should be heuristic-based scheduling.

3.2.2.2 Exclusion Criteria

- The study was not written in English.
- The publication year does not fall between 2000 and 2021.

3.3 Simulation Tools

This section discusses the simulation tools that were used in this study. The proposed method is put through tests in a simulation tool, which is used to simulate cloud computing environment. We have two options for testing this: the first is to use real-time tests like Amazon EC2 cloud services, and the second is to simulate the cloud environment using simulation tools. Due to the high expenses of directly examining, analyzing, and evaluating the performance of the proposed system using real cloud computing environment (real hardware and software resources), modeling and simulation have become a very powerful tool for dealing with these challenges. In our approach, we prefer to use a simulator because using real tests limits the studies to the infrastructure level and makes reproducibility of the results difficult. A simulation-based method decreases the complexity of the experimental setup and saves significant effort in workflow execution by allowing users to test their applications in a repeatable and controlled environment. The majority of workflow scheduling methods have been implemented in cloud simulators. Furthermore, assessing performance in a real-world cloud system is complex and time consuming. Furthermore, access to real infrastructure generates payments in real currency. In this section, we will go over some of the most popular simulation tools.

CloudSim

CloudSim has got tremendous acceptance in both academic and industry-based cloud application development. It provides a simple framework for performing experiments based on specific design concerns, which can be seamlessly merged into the offered generalized framework (Buyya et al., 2009). CloudSim was created by the University of Melbourne's GRIDS laboratory to model and simulate various cloud computing infrastructures and application services, as well as simulations of multiple data centers. It includes classes for data centers, hosts, services, users, internal data center networks, and physical host and data center energy usage. CloudSim’s architecture is made up of specific entities that are represented as Java classes. As a result, developers can easily emulate them by inheriting these classes or modifying the available methods or parameters (Bahwaireth et al., 2016).

CloudAnalyst

CloudAnalyst was developed at the University of Melbourne's GRIDS laboratory to achieve a specific set of objectives. This simulator is distinguished by its graphical output presentation

and the ability to repeat simulations to satisfy the needs of the user. As a result, when the primary purpose is to simulate distributed applications across various data centers and user groups, CloudAnalyst is favorable (Bahwaireth et al., 2016). It adopts the core features of the CloudSim framework. An easy-to-use GUI, the ability to distinguish a simulation from the programming code, quick simulation configuration, and an enhanced graphical output display with helpful formats such as tables and charts are among the new and powerful features in CloudAnalyst.

iCanCloud

iCanCloud is a cloud computing platform that is used to simulate and model large-scale experiments with a diverse set of users. Basic users are at one end of the user spectrum, while developers of big distributed systems are at the other. It can anticipate cost and performance trade-offs for a certain collection of applications operating on a single piece of hardware. SIMCAN, which is used to study high-performance I/O architectures and simulate computer networks utilizing the popular OMNET++ framework, is the foundation of the iCanCloud simulator. It also provides adaptable, scalable, and user-friendly tools. A virtual machine serves as the cornerstone of the cloud computing system in this simulator. A virtual machine (VM) can be constructed in this simulator by specifying the four basic components: CPU, memory, storage, and network. These components form the simulation engine's core, and other components, as well as component substitutes, can be uploaded to its repository. It also offers a graphical user interface (GUI) that users can use to modify their simulations (Bahwaireth et al., 2016).

CloudReport

CloudReport is another utility built on the CloudSim architecture. CloudReport, on the other hand, enhances many aspects of CloudSim, including a more user-friendly graphical user interface, the ability to conduct many simulations concurrently, and enhanced simulation results. CloudReport additionally benefits from the usage of an Application Programming Interface (API) for the development of customized extensions to emulate specific algorithms (Bahwaireth et al., 2016).

WorkflowSim

As previously noted, there are a variety of simulator environments available to simulate and model cloud computing environments and services, including CloudSim, GreenCloud,

iCanCloud, CloudSched, CloudReports, DISSECT-CF, CloudAnalyst, and others. These simulators contain similar architectures, modeling elements, and simulation processes, but they also have differences, such as an emphasis on distinct service levels and the use of different performance metrics. Although many simulators perform general activities, modeling various specialized applications such as web requests, batch processes, MapReduce tasks, and workflows still requires significant effort. CloudSim, for example, is a popular simulator for simulations of resource provisioning and job scheduling. It solely takes into account single jobs and ignores workflow task interdependence.

Scientific workflows can comprise a significant number of tasks, and executing these tasks may require the usage of a variety of complicated modules and software. Analyzing the efficiency of workflow optimization approaches in real-world infrastructures is complex and time-consuming. As a result, simulation-based studies are increasingly commonly used to assess workflow systems. The effectiveness of scheduling algorithms has been evaluated using simulators. Because users may test their applications in a repetitive, controlled setting, simulation-based methods decrease the complexity of experimental setups and save time in workflow execution environments. Despite this, a realistic simulation framework for scientific workflows is essential to obtain appropriate findings, especially as the overall system overhead plays a considerable role in the workflow's runtime.

Finally, improvements in workflow research need the development of a general-purpose framework that can accommodate widely acknowledged workflow features and optimization strategies. Existing simulators, such as CloudSim/GridSim, are unable to simulate workflows at a fine level of detail. They do not support task clustering, a popular strategy for lowering task execution overheads by aggregating small tasks into a bigger job, and they do not take overheads into account. Task clustering simulation involves two layers of execution models on both the task and job levels. A workflow-clustering engine is also required, which clusters tasks using algorithms and heuristics. These simulators also disregard other strategies like workflow partitioning and task retry.

CloudSim is a common cloud computing infrastructure and service model and simulation framework. However, it only allows for the execution of a single workload, whereas our work focuses on workflow scheduling and execution. It uses a straightforward task execution strategy

that disregards task dependencies and clustering. It also ignores the potential of failures as well as overheads. WorkflowSim[6], an upgraded version of CloudSim (Buyya et al., 2009), (Bulaja et al., 2019), was developed to satisfy these new requirements. It is developed by Weiwei Chen, a Ph.D. student at the University of Southern California, and is licensed under the Apache License 2.0. Other simulators focused on a few desired features of workflow management, such as workflow scheduling, but such simplification cannot keep up with the ever-changing world of distributed computing and the evolution of new workflow management strategies.

WorkflowSim aspires to extract the common characteristics revealed by multiple workflow systems and to support commonly used workflow management system, rather than focusing on a specific component of workflow management. WorkflowSim not only lets you test scheduling approaches, but it also accounts for overheads and faults associated with task scheduling and execution (Chen & Deelman, 2012).

Simulation Tool	**Comparison Items**					
	Programming Language	**Base Platform**	**Platform**	**Parallel Experiment**	**GUI**	**Availability**
CloudSim	Java	SimJava/ GridSim	Any	No	No	Open Source
CloudAnalyst	Java	CloudSim	Any	No	Yes	Open Source
iCanCloud	C/C++, Shell	SIMCAN	Any	No	Yes	Open Source
CloudReport	Java	CloudSim	Any	No	Yes	Open Source
WorkflowSim	Java	CloudSim	Any	No	No	Open Source

Table 2 Comparison of Different Cloud Based Simulators (Byrne et al., 2017)(Mansouri et al., 2020)

WorkflowSim (Chen & Deelman, 2012) provides workflow modeling and scheduling and it is an environment in which the proposed cloud system algorithm's credibility may be tested and simulated. WorkflowSim is an open source workflow simulator that extends workflow simulation capabilities of CloudSim tool. It supports task clustering, which combines tasks into

[6] http://workflowsim.github.io/WorkflowSim-1.0/

a cluster job, as well as a dynamic scheduling mechanism that matches jobs to worker nodes as they become idle. It includes a DAG model with support for an elaborate model of node failures, a model of delays occurring at various levels of the WMS stack, and implementations of several popular dynamic and static workflow schedulers (e.g., HEFT, Min-Min) and task clustering algorithms. To improve overall performance, users can define several criteria. CloudSim can make it easier for researchers to build up their experimental platform to explore the cloud computing simulation, whereas WorkflowSim assists researchers in optimizing their workflows based on their own requirements and parameters.

In the Grid and Cloud communities, the development of WorkflowSim has received much interest. It was applied in a range of fields such as faults, balanced task clustering, cloud brokers, energy-conscious scheduling, costs-oriented scheduling, etc. This is an environment for simulating and guaranteeing the credibility of the proposed cloud system. WorkflowSim contains a hierarchical structure. To define the hierarchy of WorkflowSim from the top to bottom, we have:

Workflow Mapper: It calls Workflow Parser to parse an XML file in order to import DAG files and other metadata information (such as file size) and generate a task list (or cloudlets in CloudSim) which consists of a user's program/activity. The Workflow Mapper then builds a list of tasks after the mapping and assign them to a site for execution. Then the Clustering Engine overtake this list from Workflow Mapper.

Clustering Engine: It minimizes scheduling overheads by merging tasks in the same pipeline into jobs (a set of activities) based on the method a user specifies (horizontal, vertical, random, or more). If this option is not selected, Clustering Engine will not merge any jobs and will instead send the list of jobs to Workflow Engine. From now on, we will have jobs, even if they only have one task.

Workflow Engine: It distributes jobs based on their dependencies, ensuring that each work can only be sent to Workflow Scheduler once all of its parent jobs have completed successfully. Only free jobs will be released to the Scheduler by the Workflow Engine.

Workflow Scheduler: Scheduler is a tool that matches jobs to worker nodes based on user-defined criteria. Both static and dynamic scheduling algorithms are supported. Jobs are assigned

to a worker node during the workflow planning stage for static algorithms. When the job arrives at the remote scheduler, it will simply wait for the assigned worker node to become available. Jobs are matched to a worker node in the remote scheduler for dynamic algorithms whenever a worker node becomes idle.

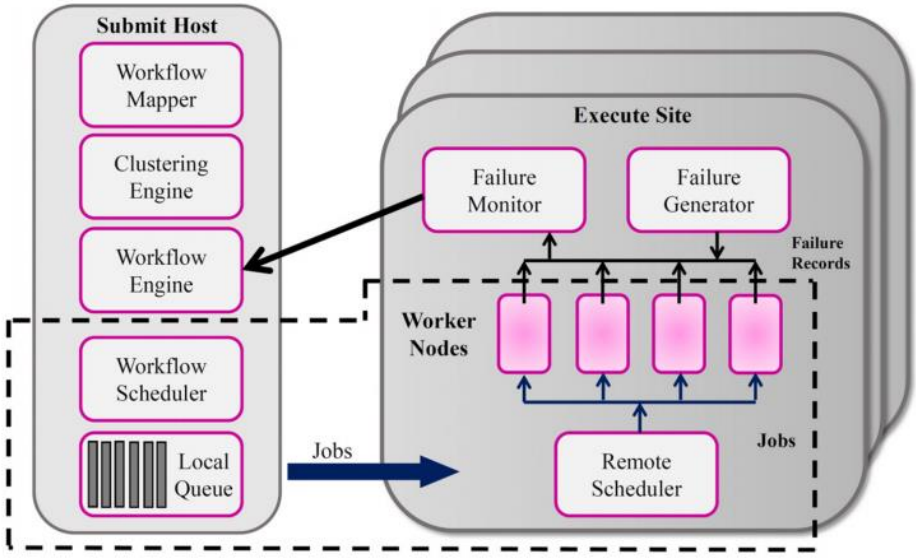


Figure 3. 1 WorkflowSim Structure: The area surrounded by dash lines is supported by CloudSim (Chen & Deelman, 2012)

The resources are divided among the cloudlets for execution for a fixed period of time in the Time Shared scheduling scheme. After this time period expires, the resources are removed from that cloudlet and assigned to another cloudlet. In this policy, the Round Robin (RR) algorithm is utilized. The resources in Space Shared are not shared among the cloudlets. The Virtual Machine is owned by a cloudlet till it completes execution. It operates on a first-come, first-served basis, thus there is a long wait (A. Kaur & Sengupta, 2017). WorkflowSim, on the other hand, has included multiple layers of overheads and failures, which increases the accuracy of simulations (Chen & Deelman, 2012).

3.4 Workflow Model

Many system models and methodologies are being developed with the goal of cloud workflow application. An application, a target computing environment, and a scheduling performance requirement make up a scheduling system model. The most widely used representation for Scientific Workflow is DAG (Direct Acyclic Graph) in which scientific application tasks are

represented with nodes and edges are for representing inter-task data dependencies. Node label appear on the graph represents computation cost of the task and edge label is for inter-task communication cost (Topcuoglu et al., 2002). Each vertex represents a task *t*, and the workflow contains *n* number of tasks i.e. $T = t_1, t_2, \ldots t_n$. Execution precedence constraints are maintained at the edges because of data dependency i.e. child tasks cannot start before all parent tasks finish their executions. DAG can be defined by (Luiz Fernando Bittencourt et al., 2010):

$$G = (T, E, C, W)$$

Where:

- **T** – The set of all tasks $T = \{t_1, t_2 \ldots t_n\}$, n is the total number of tasks.
- **E** – The set of edges between tasks $E = \{E_{ij} = (t_i, t_j) \mid t_i, t_j \in T, i < j\}$, t_i is the direct predecessor of task t_j
- **C –** The set of communication costs between tasks with edge connections; c_{ij} represents the communication costs between task t_i and task t_j
- **W** – The set of weights on the task; it represents the computation costs of the tasks.

A task t_x requires a set of files $F_x = f_1, f_2, \ldots f_n$ to be staged in order to be executed. Each file f_x requested by a task t_x can be hosted at different sites on data storages. Resource is a set of compute and data resources and can be represented as $R = \{r_1, r_2, \ldots, r_n\}$. A set of partial segments of each file $d_x \subset f_x$ that must be transferred for a task t_x^r given to a resource r is denoted by: $data\_set = \{\{d_x \rightarrow r\}^{t_x^r} \forall d_x \sqsubset f_x, r \in R, |\{d_x\}| \leq |R|\}|$. In a set containing several tasks and files, the index x provides a relationship between the set of files that the task requires and the task itself. The set of immediate predecessor tasks of a task t_i is defined as $pred(t_i) = \{t_j \in T \mid \exists f \in F, \text{such that } (t_j, f) \in E \wedge (f, t_i) \in E\}$. Similarly, the set of immediate successors is given by $succ(t_i) = \{t_j \in T \mid \exists f \in F, \text{such that } (t_i, f) \in E \wedge (f, t_j) \in E\}$ where E stands for edge, the set of precedence relation between tasks and data files and $t_i, t_j \in T$.

Task execution cost represents the execution time of task t_i on virtual machine q_j, and given by the following equation (Topcuoglu et al., 2002):

$$w_{i,j} = \frac{t_{length(i,j)}}{q_{comp(i,j)}}$$

Eq. 1

Where $t_{length(i,j)}$ - indicates the task required to execute the instruction length, $q_{comp(i,j)}$ - represents the computation ability of the virtual machine *j*.

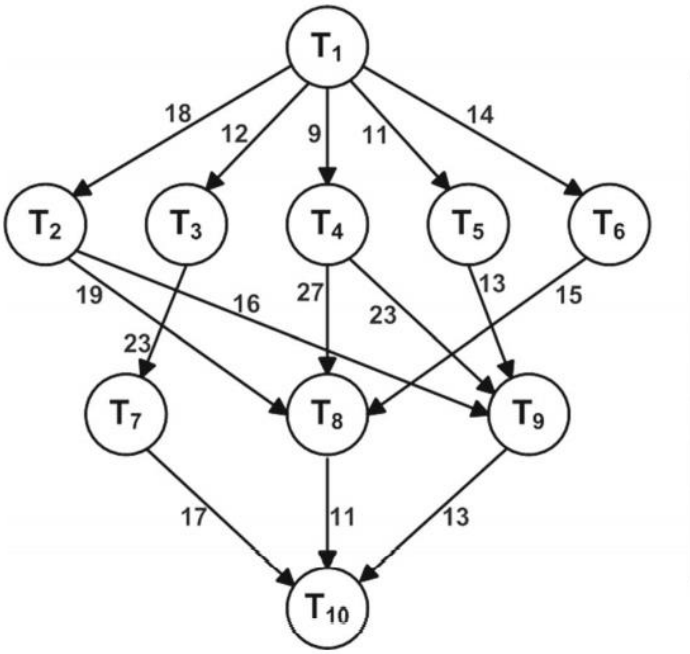
Computational performance of the virtual machine *j* can be calculated according to the following equation (Topcuoglu et al., 2002):

$$q_{comp(i,j)} = q_{pesNumber(i,j)} * q_{mips(i,j)}$$

Eq. 2

Where $q_{pesNumber\ (i,\ j)}$ - the number of processing cores of virtual machine *j*, $q_{mips(i,\ j)}$ - machine processing capability of virtual machine *j*.

Figure 3.2 shows an example of a directed acyclic graph (DAG) with ten tasks and a heterogeneous system with three processors. In addition, we assume that each processor in the heterogeneous system can compute and communicate at the same time. The task node number is represented by the label in the circle (node), and the number next to each node represents the mean computation costs of the tasks across all processors. Although there are numerous cost models available, the one provided in this chapter is made up of time cost, or execution time, and cost for the execution of workflows in clouds, which are the most common goals. This cost model will be applied in the provisioning of virtual machines as well as the activation scheduling. A cost model is a set of formulas used to predict the cost of executing operations in accordance with a schedule (de Oliveira et al., 2019).



Task	P_1	P_2	P_3
T_1	14	16	9
T_2	13	19	18
T_3	11	13	19
T_4	13	8	17
T_5	12	13	10
T_6	13	16	9
T_7	7	15	11
T_8	5	11	14
T_9	18	12	20
T_{10}	21	7	16

Figure 3. 2 Simple DAG application model example with 10 tasks and the Estimated Computation Cost, $w_{i,j}$, of the tasks on 3 processors [4]

The average computation cost (ACC) is $v \times q$ computation cost matrix in which each $w_{i,j}$ represents the predicted execution time for task t_i on processor q_j. A task's average computation cost is calculated as follows (Topcuoglu et al., 2002):

$$\overline{\mathrm{ACC}_\mathrm{i}} = \sum_{j=1}^{q} w_{i,j}/q$$

Eq. 3

Where $\overline{\mathrm{ACC}_\mathrm{i}}$ – average computation cost of task t_i, $w_{i,j}$ – the estimated execution time to complete task t_i on processor q_j , q – the total number of processors.

The data transfer rates between processors are stored in matrix B of size $q \times q$. The communication startup costs of processors are given in a q-dimensional vector L. The communication cost between tasks t_i, which is scheduled on q_m and t_j that is scheduled on q_n are given by (Topcuoglu et al., 2002):

$$c_{i,j} = L_m + \frac{f_{i,j}}{B_{m,n}}$$

Eq. 4

Where L_m – the communication startup time of the processor q_m, $f_{i,j}$ – the data file transferred from task t_i to task t_j, $B_{m,n}$ – the data transfer rate between the processors q_m and q_n.

The communication cost is zero when two tasks, t_i, t_j are assigned to the same processor. Otherwise, data must be transferred from the processor on which task t_i is assigned to the processor where task t_j is assigned. When both tasks t_i and t_j are mapped to the same processor, the expected communication time, $c_{i,j}$, becomes zero because it is assumed that the inter-processor communication cost is negligible. Before scheduling process starts, each edges of DAG must have labeled with the mean communication costs. The average communication cost of an edge *(i, j)* is defined by (Topcuoglu et al., 2002):

$$\overline{c_{i,j}} = \bar{L} + \frac{f_{i,j}}{\bar{B}}$$

Eq. 5

Where $\bar{B}$ – the average transfer rate among the processors, $\bar{L}$ – the average communication start-up time of all processors.

For this study, we assumed the value of B is the same among all processors and L is set to zero. The communication cost exists when two interdependent tasks are scheduled to two different processors. Nevertheless, if they are assigned to the same processors, the expected communication time regarded as zero. Therefore, in this research paper eq. (4) is equal to eq. (5).

To execute workflow, the SWfMS must first set up the appropriate execution environment and then run the applications in virtual machines. Initialization is the process of starting the virtual machine, installing packages, and configuring virtual machine parameters for using it to carry out the executions of the workflow. The provision of IaaS is accomplished by the instantiation of virtual machines (VMs). VM deploying is the action of creating it under a user account in the cloud. Type and location of the virtual computer, i.e., the cloud site, are determined by its deployment. The earliest execution start time EST(t_i, q_j) and earliest execution finish time EFT(t_i, q_j) of task t_i on processor q_j for the entry task t_{entry} can be calculated with (Topcuoglu et al., 2002):

$$EST(t_{entry}) = 0$$

Eq. 6

EST and EFT values are computed recursively for other tasks in the task graph, starting from the t_{entry}. In order to compute the $EFT(t_i)$, all immediate predecessor of task t_i must have been scheduled by taking into account the transfer time.

$$EST(t_i, p_k) = \max(\, max_{t_q \,\epsilon\, pred(t_i)} \left(FT\left(t_q\right) + \, c_{q,i}\right), AT(p_k))$$

Eq. 7

Where $pred(t_i)$ – the immediate predecessor tasks of task t_i, $FT(t_q)$ – the finish time of task t_q, $AT(p_k)$ – the earliest available time for the processor p_k, the communication cost $c_{q,i}$ is zero if the predecessor node t_i is assigned to processor p_k.

Similarly, the $EFT(t_i)$ on processor p_k is defined by (Topcuoglu et al., 2002):

$$EFT(t_i, p_k) = \, w_{i,k} + EST(t_i, p_k)$$

Eq. 8

To summarize, the task-scheduling problem is to allocate tasks to processors in the most efficient way feasible so that all of the aforementioned requirements are met while also having the shortest overall execution time (makespan). The makespan also refers to the overall duration of the schedule. That is, the makespan is equal to the exit node in the DAG's maximum Actual Finish Time (AFT). As a result, makespan is defined as (Topcuoglu et al., 2002):

$$makespan = \max\{FT(t_{exit})\}$$

Eq. 9

Where FT is the finishing time of the exit task.

Upward task of task graph, $rank_u$, is calculated starting from the exit task to upward recursively. To prioritize the $rank_u$ value of the task, the calculation time of the task, the communication time between the tasks and the influence of the priority of the subtask on the priority of the parent task is taken into account. A task with a higher $rank_u$ value has a higher priority and the

earlier this task is assigned to the processor. When tasks have same upward ranks, the priority among tasks is decided randomly [3]. The upward task of every non-exit task is calculated by:

$$rank_u(t_i) = \overline{w}_i + max_{t_j \epsilon succ(t_i)}\left(\bar{c}_{i,j} + rank_u(t_j)\right)$$

Eq. 10

Where $\overline{w}_i$ – the average computational cost of task t_i, *succ(t_i)* – the set of immediate successors of task t_i, $\bar{c}_{i,j}$ – the average communication cost of edge between task t_i and t_j

Moreover, to calculate the upward rank $rank_u$ of exit task t_{exit} we use:

$$rank_u(t_{exit}) = \overline{w}_{exit}$$

Eq. 11

3.5 Virtual Machines in Cloud Computing

Data centers have expanded exponentially in recent years, and all major IT businesses are investing their money into the creation of new data centers. Data-center automation ensures that large numbers of hardware, software and database infrastructure in these data centers can be automatically distributed to millions of Internet users at the same time, with assured QoS and cost-effectiveness. Companies like Google, Yahoo!, Amazon, Microsoft, HP, Apple, IBM, and others are all in the game. Many of these organizations have spent billions of dollars in data center design and automation (Hwang et al., 2012).

A large number of heterogeneous workloads can be executed on servers at different times in data centers. These heterogeneous workloads can be loosely classified into two categories: chatty workloads and non-interactive workloads. Chatty workloads will burst at some stage and go back to a silent state at some other point. An example of this is a web video service, where a lot of users use it at night and few people use it during the day. Non-interactive workloads do not require individuals to make progress after they have been submitted. High-performance computing is a common example. The criteria for the capital of these workloads are drastically different at different levels. However, in order to ensure that the workload will still be able to deal with all forms of demand, the workload is statically distributed enough capacity to meet the peak demand. For this case, the granularity of resource management depends on CPU, memory, and network interfaces (Varghese et al., 2016).

Cloud computing is powered by data centers which is designed to house numerous servers (a single data center can easily contain 10,000 racks with 100 cores in each rack, i.e. 1,000,000 cores total), communications and storage systems co-located in a massive networked-format for a reason of their similar environmental, physical protection and maintenance requirements. Data center generally provides utilities needed such as power, cooling system, shelter, and security. The PMs which are owned by CSPs are white-box resources: the CSP knows precise information on their status (i.e. electricity consumption, present working load, temperature), and the CSP's responsibility is to maximize their utilization. These PMs may reside in one or more DCs. When two PMs which are located in different DCs are instantiated, communication latencies between them are often longer than when both PMs are located in the same DC (Mann, 2015).

When it comes to provisioning resources, virtual machines (VMs) are the fundamental building blocks of the cloud's infrastructure-as-a-service (IaaS). Upon submission, a user-defined job is split into a number of tasks, with each task being assigned to a single virtual machine (W. Lin et al., 2016). The degree to which a PM's services are consumed by the VMs residing therein is determined by its usage. In terms of use, the CPU is the main resource. It is in the CSP's interest, that the CPU should be used in a way that enables maximum utilization to be made of the resources available. However, if the CPU load is too high, the VM living on the PM is unlikely to get the power needed to break the SLA and affect customer satisfaction. Too heavy CPU load can overheat and speed up hardware aging. Many researchers focused on the CPU loading for this reason. Other resources, such as memory or disk space, can also create a bottleneck. Cache is of special significance because existing virtualization solutions do not provide the isolation of the cache use of various VMs accommodated by the same PM, resulting in conflicts. As a result, it is critical to model and predict the performance interference that can occur when two virtual machines are co-located (Mann, 2015).

The main target for virtualization continues to be on servers today. Amazon's Elastic Compute Cloud (EC2) is a typical example of a web service that offers elastic processing resources in the cloud. EC2 helps users to set up VMs and maintain user accounts for the time span of their use. A server machine can be partitioned into several Virtual Machines (VMs) each offering an independent server environment large enough to support a single application or parts of it in a secure and resource-proof manner. The resource allocation to VM can be modified at runtime to dynamically balance the workload specifications of the virtualized program.

Typically, the VM is defined by the following:

- Number of cores of the CPU
- CPU processing capacity per core (e.g. one million instructions per second (MIPS))
- Size of RAM (e.g., in gigabytes)
- Size of disk (e.g., in gigabytes)

There may also be communication (bandwidth, latency) specifications between pairs of VMs or between VMs and the customer (Mann, 2015).

CSPs can provide VMs in two possible ways: either the CSP predefined certain VM configurations from which customers can select (e.g. Amazon EC2) or customers can define their own VM configuration by defining the appropriate amount from each resource (e.g. IC Cloud); this can make a difference in the efficiency achieved (Mann, 2015). VMs should indeed be ranked in order of their efficiency and cost effectiveness so that the user can deploy an application on a cloud VM that maximizes performance (Varghese et al., 2016). The user shall have as input a set of weights defining the memory and process, local communications, computation, and storage specifications of the scientific application to be implemented in the cloud. The weights along with the attributes of each VM are used to determine a score resulting in a VM rating. Two sets of ranks can be generated; the first ranking is focused solely on the performance of the VMs and the second ranking is based on both performance and cost. From the first set of rankings, we presume that the highest ranked VMs can maximize application efficiency on the cloud, and the highest ranks from the second set can enhance application performance in a cost-effective manner.

Data mapping on storage nodes is a strongly connected concern. Large amounts of data are mapped to specialized storage nodes in some systems, resulting in high network traffic between computation and storage nodes. In such cases, the placement of virtual machines on computing nodes and the placement of application data on storage nodes are two interrelated considerations that must be taken into account jointly in order to avoid an excessively large network. There is communication with entities outside the cloud in addition to communication between VMs and storage nodes. The users are a good example of this. The reaction time experienced by users is critical in a variety of applications. Response time is the sum of network round trip time and

processing time, and it can be reduced by delivering user requests from DC, which offers a combination of low latency and quick processing.

Standard metrics for measuring cloud VM attributes as closely as possible to the underlying hardware depend on evaluating attributes that are closer to hardware, such as frequencies (number of operations in seconds), latencies (microseconds or nanoseconds) and bandwidth. Cloud virtual cluster performance estimation can be done in many different methods. This study profiles a VM based on five main attributes: MIPS, RAM, processor cores, bandwidth and storage capacity. It is possible to directly derive the values of these properties from specified VM attributes (W. Lin et al., 2016). In order for the VM to be ranked precisely based on their performance, the value of the attributes are needed to be normalized (Varghese et al., 2016). This way the view of the relative performance for each attributes of VM can be provided. For instance, a VM that is well ranked in the memory may poorly perform in the storage. Obviously, some attributes are more decisive and important than the other. During the calculation of ranking values of VM, such attributes take a huge portion. After calculation of the value of attributes, a higher score denotes a higher rank. Eventually, the generated scores, based on both performances are ordered in a decreasing order to produce the final score, which is the ranking of the VMs.

Price and pricing schemes can vary significantly between CSPs, and even the same CSP may provide multiple pricing schemes. Some providers, for example, provide lower long-term rental prices and higher pay-as-you-go rates. The latter is equally concerned with the hour's quantum time. In addition, a fee may be proportional to the amount of time that particular services, such as network bandwidth, are used. In recent years, a new pricing structure has emerged; spot occurrences, the price that is determined by the provider's present pressure. Spot instances are inexpensive when the provider has a lot of free space, but they become more expensive as the provider's load grows (Mann, 2015).

Let us consider the Amazon EC2 cloud service. The Amazon Elastic Compute Cloud (EC2) platform is open to the public and offers a variety of virtual machines (VMs) with varying performance capabilities. It is a cloud-based web service that delivers secure, resizable computational capacity. General purpose, memory optimized, cluster compute optimized,

storage optimized, and GPU instances are some of the previous generation EC2 VMs, also known as instances. VM Performance Metrics

Various ways could be used to calculate performance of VM, however, we used MIPS, RAM, Storage, Processing Entity, and Bandwidth as main score calculation criteria.

3.5.1 MIPS

Processor (CPU) has a significant impact on the speed with which programs load and run. There are multiple ways to measure processor speed like clock speed (frequency), mips (millions of instruction per second), flops (floating-point operations per second)…etc. MIPS rating is used as the general measurement or benchmark of how many instructions a processor can handle in a single second. We used MIPS to calculate the processing speed of the machines. MIPS is calculated by dividing an application's instruction count by its execution time*10^6 (O'Loughlin & Gillam, 2014).

3.5.2 RAM

We used RAM performance to compute capacity of main memories in gigabytes (GB), not other issues related to I/O performance like latency or throughput. Because memory is a major cost driver and the most commonly constrained resource in virtual and cloud environments, properly sizing memory is critical. Large amounts of RAM, like the CPU, are required if VM is expected to execute task very quickly. The amount of RAM installed on your server determines how many processes the CPU can handle at any given time. More RAM means the CPU has direct access to more data and instructions.

3.5.3 Processing Cores (Processing Elements)

The number of cores in a processor determines how well the system can handle multiple tasks at once, i.e. computational ability. Cloud resource can be provides machines with different number of processing cores or processing elements to improve processing speed by using two or more processors that can run independently or cooperatively. CPU cores can run applications or threads on their own. Many cores can exist on a single CPU. Some programs can split work between cores, giving multi-core CPUs a speed boost even when they are not multitasking.

3.5.4 Bandwidth Strength

To evaluate network performance we used bandwidth. Bandwidth is a network's maximum transfer throughput capacity. Bandwidth is the amount of data that can be transferred between the server and the user in a given amount of time and it can be measured in bits per second, megabits per second (Mbps), or gigabits per second (Gbps). To free up bandwidth for other servers, CSP may limit the bandwidth for a specific server(s). This throttling usually occurs only during peak hours, when a large amount of bandwidth is being consumed due to the amount of traffic received or data requests made on a specific server or servers. If your server's bandwidth is regularly restrained, it can significantly reduce the performance of your cloud server because your server will not be able to transfer data quickly enough to meet the requests the server is receiving. High bandwidth can affect the overall performance of all the applications. When two PMs are located in different DCs, communication latencies between them are typically longer than when both PMs are located in the same DC. Furthermore, in our current work, we do not consider network latency or read/write operation latency when calculating performance metrics, but we have observed the following situation, which has a significant impact on performance.

3.5.5 Storage Capacity

In typical scientific workflow, which deployed on high-performance computing facilities like clusters and grids, due to a storage constraint, the scientists must delete all intermediate data. For this reason, to avoid this storage bottleneck cloud is the leading option. Theoretically, the system can provide unlimited storage resources in cloud for fee. Therefore, we can store all of the intermediate data generated during execution. Different CPs offer block storage services that integrate with their respective VM compute services, with a variety of block storage types that can be configured for different levels of performance and pricing. In cloud, VM consists of locally attached disk that is physically connected to the machine that is running your instance. For instance, Amazon EC2 provides up to 2500 GB per volume which can be varies by instance type, while Google cloud provides 356 GB per volume (Edward Jones, n.d.).

CHAPTER FOUR

PROPOSED SYSTEM DESIGN

4.1 Design Considerations

This section focuses on developing a novel scheduling strategy for data-intensive scientific workflows, which can result in better resource allocation and job dispatching. It is primarily concerned with scheduling data-intensive scientific workflows to reduce total data transfer time and/or cost, storage space used, total execution time, and/or a combination of these factors. The acquisition and allocation of appropriate computing resources, as well as the mapping of task execution on those distributed resources, are all part of the scientific workflow application scheduling process. Many factors, such as scheduling policy, load balancing, computational time, data locality and data transfer, fault tolerance, and QoS were taken into the consideration during the design of this proposed system and are summarized below.

4.1.1 Resource Utilization

Energy consumption of underutilized resources accounts for a significant portion of total energy consumption, especially within the case of the cloud computing environment. Moreover, a resource allocation strategy that considers resource utilization improves energy efficiency. This extends further in the cloud with virtualization technologies, allowing tasks to be easily consolidated. Task consolidation is a mechanism of reducing the total number of resources used by maximizing the utilization of active resources. Consolidating tasks is a good way to get more out of your resources. It also cuts down on energy usage. In addition to this, resource prioritization is used, which has a direct consequence on resource utilization.

4.1.2 Scheduling Policy

Scheduling policies are algorithms for allocating the CPU resources to the simultaneous tasks that are deployed (i.e. assigned to) a processor or a shared processor pool. Efficient scheduling policy uses the perfect resources for matchmaking techniques based on the job requirement. Always try using fewer storage devices based on the cloud user's job requirement and it continuously saves computing time and space (.R & .T, 2018). The makespan and overall cost of a workflow are determined by the data sources used and the task-to-resource mapping. When the size of data is comparatively larger than the computation time of tasks, selecting the "best"

data sources before ignoring compute sources, as is done in existing scheduling algorithms, which does not result in time and cost efficient schedules. Prior to the execution of the tasks started in a workflow, significant bandwidth is required to stage-in and stage-out these data. If the data is to be reused, the scheduling policy must choose compute resources that are closer (in terms of network distance) because this affects the time/cost of transferring output data and if data-host are closer in terms of network distance, the transfer time is considerably reduced.

4.1.3 Data Transfer

Workflows are loosely-coupled parallel applications that consist of a set of discrete computational tasks logically connected via data- and control-flow dependencies. Unlike tightly-coupled applications in which tasks communicate by sending message via the cluster network, workflow tasks typically communicate using files, where each task produces one or more output files that become input files to other tasks. These files are passed between tasks either through a shared storage system or using some file transfer mechanism. In order to achieve good performance, these storage systems must scale well to handle data from multiple workflow tasks running in parallel on separate nodes (Juve & Deelman, 2010).

In the cloud environment, the location where the computation occurs can be different from where the input data to the workflow is stored. It is therefore, if only the computational cost is taken into account in the scheduling process, the benefits may be counterbalanced by a high data access cost. There are applications involving numerous, parallel tasks that both access and generate large data sets, sometimes in the petabytes like in high-energy physics, bioinformatics, and other disciplines. Large data sets needs specialized storage and management systems. Despite that, there is a problem with the computation-data separation in the cloud. This factor influences the computational node chosen for a task: if data transfer costs are low, some tasks can be completed faster on one computational node, but if data transfer costs are high, another computational node may be preferred. Although scheduling algorithms in traditional computing paradigms barely consider the data transfer problem when mapping computational tasks, in the cloud, this omission will be costly. Only a few papers in the current literature take into account the optimization of computation and data transfer scheduling in tandem.

Before any data-related task can be executed at a compute resource, the data must first be staged there. Output data may be similar in size to the input data and is produced at the end of or during

execution. These intermediate data should be stored in case they are needed for successor tasks. Depending on the policy for retaining or deleting the output data, the sites where the output data are stored could be potential data sources. As intermediate output files are stored, the total number of data-hosts grows. The choice of data hosts becomes more difficult as a result of this. These tasks' computation requirements cannot be denied. The tasks must then be assigned to resources for execution after the set of candidate data-hosts has been verified. The objective function determines how tasks are assigned to compute hosts. The workflow task scheduling primarily focuses on one or more of the following objective functions: minimizing the total makespan, minimizing the overall cost (economic cost) of execution and data transfer, and executing within the deadline and allocated budget. A complex sub-problem of the general task scheduling problem is mapping workflow tasks to minimize one of the objective functions.

We proposed to design the whole interaction of components, in particular the interactions between the workflow scheduling and data movement components, as a context related to minimizing data movement. Clearly, this data movement aware scheduling strategy is capable of decreasing data transmissions. We focused data locality, i.e. on moving computation rather than data to save bandwidth to some extent. In our context, by data locality we mean that instead of moving big-size data, moving the computation to where the actual resides on the node assigning the successor tasks with huge data size to the same VMs as their predecessors as much as possible. This reduces network congestion and data movement while increasing overall system throughput by means of mapping workflow tasks across computing resources based on an analysis of the entire task graph’s dependencies with the intention of completing the execution as quickly as possible.

4.1.4 Fault Tolerance

The architecture incorporates a virtual memory segment with a direct byte-for-byte relationship to a portion of the resource. Before the system fails, this memory mapped file will be used to record the status of tasks that need to be processed, are being processed, and have been completed. The persistence model is also used to safeguard the scheduler from failure during resource dispatching. The state of the algorithm is saved in the database on a regular basis. As a result, the scheduler can pick up where it left off. When performance fluctuation is detected, rescheduling is also used to migrate scheduled tasks to alternative resources. When the scheduler

receives a performance change event from the executor at run-time in the static initial scheduling, it reallocates pending tasks according to the earliest-finish-time policy.

Unless an efficient fault tolerance mechanism is in place, if a task fails, the delay in execution affects the starting time of all other dependent tasks in the workflow. The majority of time in a data-intensive application is spent communicating data between tasks and resources. As a result, there's a high possibility that something will go wrong during this stage. When a task fails to execute, the task is either migrated to another host or re-inserted to be executed on the same host, depending on the scheduling policy. The migrated tasks may fail again at the new site if data transfer occurs due to rescheduling. Before bringing the task to the site, it is necessary to make an informed decision based on its previous execution, the current state of available resources, and reliability concerns. As a result, our work focuses on mapping tasks on the reliable resource which we thought from the resource's profiling the more performant resource, the more reliable it is.

4.1.5 QoS

Quality-of-Service (QoS) management, which is the problem of allocating resources to the application to guarantee a service level along dimensions such as performance, availability, and reliability, is one of the challenges posed by cloud scheduling algorithms (Ardagna et al., 2014). When scheduling workflows, users' Quality of Service (QoS) factors such as budget (the cost of using the services) and deadline (the time it takes to complete the application) should be considered. For cloud users, who expect providers to deliver the advertised quality characteristics, and for cloud providers, who must balance QoS levels with operational costs, QoS is critical.

4.2 Description of the Proposed Solution

Workflows are abstractions that are used to model a logical flow of multiple scientific applications that can be run locally or on distributed system. Thus every workflow is made up of multiple tasks or activities that are required to produce a specific experiment's result. Each activity may consist of a number of different tasks that can be completed. Each task is the smallest processing unit, and it can run in parallel by consuming a different portion of the input data or parameter values. The workflow is made up of a few activities from the scientist's perspective because the scientist only thinks of the abstract representation of the workflow. Each

workflow, on the other hand, may be made up of thousands or even millions of tasks when it comes to execution. We consider from the execution point of view because the representation of the workflow proposed in this paper is focused on scheduling workflow tasks in clouds.

4.3 Cloud Model and Assumptions

Computer resources are implemented as virtual machines (VMs) in the IaaS (Infrastructure-as-a-Service) cloud. VMs come in a variety of computational performance, depending on the types of VMs provided by the CSP. Each VM is defined in terms of its computing power and communication capacity. We assume that we can estimate the computing capacity of each VM type in millions of instruction per second (MIPS) as it is used as a computational metrics in many cloud simulators like CloudSim and WorkflowSim. As a result, we can estimate how long a task will take to complete on a specific VM type. Likewise, the computing time of a task can be also defined based on its size and the VM type.

We also assumed that an IaaS system consists of a set of self-contained virtual machines (VMs) running on a host or physical machine in a different data centers. All of these VMs are governed by a global cloud manager who is either inside or outside the cloud data center. The cloud scheduler is integrated into the cloud manager to track the status of the deployed VMs. The user sends a workflow application to the cloud manager via SWfMS, which consists of several tasks. It then divides the workflow application into supporting tasks, organizes them by priority, and distributes them to the virtual machines. Another assumption is that the cost of communication between two interdependent tasks assigned to the same or different VMs on the same physical server is zero or negligible. The host properties are configured to support the IaaS capability model. The infrastructure parameters for service level agreements between users and cloud service providers are storage, computing, and network resources at the host level. We also assume in this work that all VMs are deployed prior to the workflow execution. Although clouds traditionally provide on demand VMs, the majority of providers also have reserved VMs available for use. Below is the list of our proposal assumptions.

a) VM is already provided by the provisioning system based on the user input and QoS parameters
b) Each task is executed sequentially in its own virtual machine (no preemption is allowed);
c) Static data files have been pre-allocated in virtual machines;

d) There is enough space in the cloud environment to store static and dynamic data files, i.e. the cloud environment's total storage capacity is greater than or equal to the sum of the size of all data files used by the workflow;
e) Writing and reading operations of data files are the only means of communication between tasks in the workflow;
f) Multi-source data transfer;
g) Target computing environment is fully connected heterogeneous processors;
h) The inter-processor communication is assumed to be free of contention;

4.4 Proposed System Architecture

This section, the architectural model, specifies the target architecture's main features. So far, a number of scheduling algorithms have been proposed that have been derived from other preexisting approaches that are specifically designed to solve provisioning problems in that time and scenario. Scheduling algorithms, for example, that we have been using in various computing environments, such as operating systems for Personal Computers (PCs), have proven to be very promising and yield optimized resource usage. These schedulers could be considered major contributors to the development of today's high-performance scheduling techniques.

Even so, this does not necessarily imply that they are appropriate algorithms for cloud computing systems, which are relatively high-scale performance computing systems. Cloud computing differs from other computing systems in its nature, particularly in terms of resource provisioning demand. The frequent variation of users, as well as the frequent variation of resource demand in a cloud environment, are two of the most important factors. Aside from that, the “Pay-as-you-go” cloud resource provisioning model is another factor that makes a significant difference.

Many cloud scheduling methods calculate the cost of executing activations on available virtual machines first. Then, after storing all alternative schedules and analyzing all activations, the schedule with the lowest cost is picked. After that, the scheduling plan is revised, and the activation is carried out. Finally, the available activations and idle virtual machines listings have been updated. The final scheduling plan is delivered at the end of the algorithm.

The cloud is a promising infrastructure for workflow execution since it provides a variety of computer resources and essentially endless computing power. Cloud providers give virtual

machines to the general public, including scientists, through Infrastructure-as-a-Service, or IaaS. Virtual machines come in a variety of shapes and sizes. The virtual machine type determines some parameters, including as the number of virtual CPUs, RAM capacity, and hard disk default storage size. Virtual machines that have been deployed in the cloud can be used to form a virtual cluster. However, because this estimation and scheduling require numerous characteristics, such as virtual machines, the problem of identifying the number and kind of virtual machines to deploy, as well as scheduling all virtual machine activations, remains open for workflow execution in the cloud (de Oliveira et al., 2019).

We assumed a single entry node and a single exit node for an input DAG. As a result, if a DAG has more than one single entry or exit node, the two nodes are connected to an extra imaginary parent or child node, respectively, i.e., to a zero-cost pseudo entry and/or exit task with zero-cost edges, which has no effect on the scheduling process. Mainly, the scheduling techniques, proposed in related works, do not consider that the effects of the rank of the tasks after the immediate tasks in the critical path. Most of the works used either upward rank or downward rank, which treats the rank of the task in one direction only. Opposite of this, the proposed scheduling algorithm adopts effective ranking strategy with communication cost problem at the center.

Score of VM defines the capability of virtual machines and the score concept can be used to assess the capabilities of cloud hardware resources, as it represents the system's minimum performance. Score-based scheduling enables us to schedule workflow applications on high-performance, reliable machines, resulting in fewer workflow application failures. Although it can causes power consumption problem, to improve the overall performance of the data centers, assigning tasks to high performance servers may be effective (W. Lin et al., 2016). Only those machines that meet a minimum threshold of workflow tasks are scheduled using the score-based workflow scheduling algorithm (R. Singh & Singh, 2013). According to their capacity, different resources have different performance based on their MIPS, RAM, Storage, Processing Entity, and Bandwidth. The Scoring System is a component that evaluates and ranks the provided virtual machines (VMs) based on the listed attributes. Schedules with a reputation system are combined in score-based scheduling. As a result, this scheduling algorithm can select high-performance resources, thereby improving cloud reliability and performance. This approach is focuses on punishing volatile, selfish, or malicious resources (for example, exclusion). As a

result, score-based scheduling motivates volunteers to donate their time and resources eagerly and consistently.

The proposed algorithm finds the resources with the highest score that will satisfy the minimum task threshold and accomplish the work in the shortest time possible at a reasonable cost. Each hardware component is assigned a score, and the component with the maximum sub-score has a high possibility of influencing the overall score of the system. The final score shows the machine's minimum performance based on the capabilities of the various machine parts. Additionally, scoring is essential for optimal job consolidation on the specified host. Data centers are made up of a set number of physical servers, each with a multi-core architecture, varying RAM sizes, different bandwidth strength, and storage size. These machines consists of different performance based on their performance of hardware components.

Moreover, thorough analyses of various cloud-related literatures, and resource provisioning models in particular, demonstrated that the absence of some key components inside cloud schedulers has a negative impact on resource provisioning efficiency. The cloud environment requires a scheduler to efficiently utilize the given resource by the resource provisioning component. This scheduler deals with the unique behaviors of the cloud in general and its resources in particular. To overcome this issue, this paper introduces server score-based scientific workflow scheduling, which consists of five major components, as illustrated in *fig 4.1*.

The system is separated into two primary portions to demonstrate the system architecture from the top to bottom. In the first layer of the system is the user, as well as the components with which the user interacts. Users can be scientists, academicians, researchers, entrepreneurs, government agencies, or anybody else who wishes to rent cloud computing services to run any business workflow with the help of SWfMSs. SWfMS is a piece of software that allows users to run workflows on distributed systems that can perform executions of different sized. It handles process execution logistics and makes judgments on resource selection, data management, and computation scheduling. SWfMS allows access to storage services to store workflow data, network monitoring services for determining network distances, data registry services for monitoring the workflow data location, and workflow computing services to execute workflow tasks (Casanova et al., 2018).

SWfMSs can be installed directly on the virtual machines and administer services delivered in the cloud in a single-site cloud environment. To run workflows in the cloud, SWfMSs can use certain middleware to deploy or remove VMs and facilitate communications between VMs. Coasters in Swift/T, Kepler EC2 actors, CloudMan in Galaxy, and RabbitMQ12 in Triana4 SWfMS are examples of such middleware. For scientists, SWfMS offers a variety of advantages including scheduling strategies, fault- tolerance and provenance management mechanism (de Oliveira et al., 2019). Another entity we found in the first layer of this architecture is stage-in/stage-out data. Before individual tasks execute, the inputs of data-intensive computational workflows are typically staged into computational resources. Furthermore, data sets that are generated by a workflow are often staged off a compute node and onto storage systems after workflow execution is completed.

Additional data transfer activities may be necessary to transmit intermediate data products from the nodes where they are generated to the nodes where they will be consumed by other tasks in the workflow. If a node's storage space is insufficient, some stage out operations may be required before subsequent computations can execute on that node. Pegasus workflow planners, for example, explicitly generate data staging tasks to move data on and off computational nodes before and after task execution while ensuring proper control flow dependencies. Therefore, data staging and transfer procedures are often included in workflow planning and are submitted, run, and monitored by the workflow management system in the same way as computational jobs are (Bharathi & Chervenak, 2009). In some cases, the input data of a workflow may be spread over multiple sites and cannot be relocated. To enable workflow execution in a multi-site cloud with this distributed input data, the activations of each activity must be performed at separate locations (de Oliveira et al., 2019).

After user submit the workflow, cloud resources are provided to execute it. The resource center depicted in the first layer of the architecture represents the overall characteristics of cloud computing. This environment shows the overall properties of the cloud as a whole but, specifically service model, IaaS. These resource centers used by users can becloud approach from a single cloud provider or multi-cloud environment. This component simulates several nodes that make up the cloud's resources. Each host in a resource center houses one or more virtual machines. Therefore, the aforementioned diagram of resource center encompasses Servers, Hosts, and Virtual Machines in each Host.

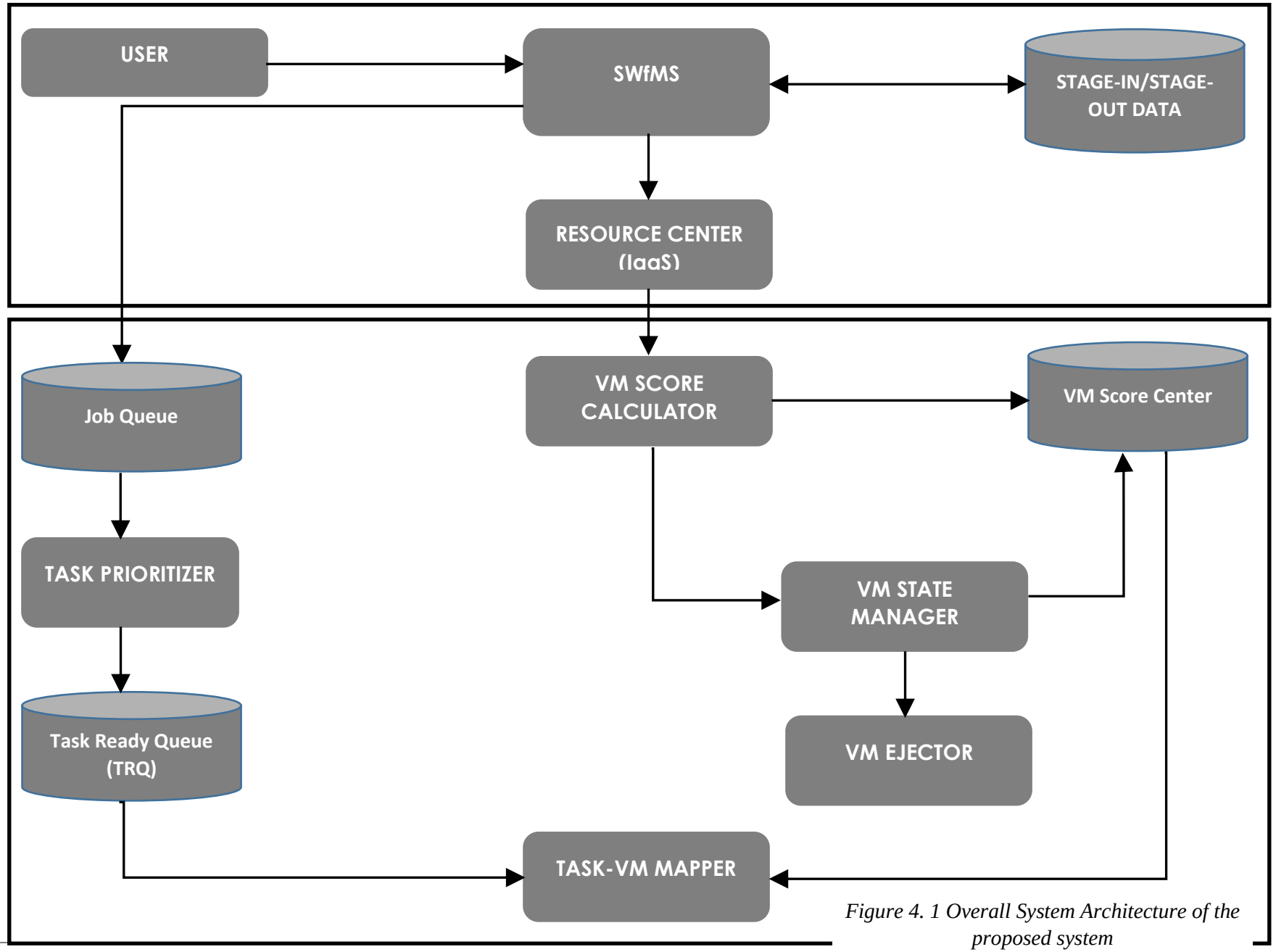


Figure 4. 1 Overall System Architecture of the proposed system

The main components of the proposed model comprise the second part of the system architecture. The new model is structured in such a way that each resource center contains one additional embedded system known as a VM score calculator. Each component is described in detail below.

4.4.1 VM Score Calculator

The score calculation procedure considers a variety of QoS parameters, including MIPS, RAM, Storage, Processing Entity, and Bandwidth. According to their operations, resources have different performance level of MIPS, RAM, Storage, Processing Entity, and Bandwidth. A scoring mechanism, therefore, is required to rank VM capacity. Schedules are combined with a reputation system in score-based scheduling. As a result, our scheduling technique may select high-performing resources, so that it can enhances reliability and overall cloud performance. Moreover, scoring is important to consolidate work on the particular VM efficiently. A VM with the highest score will be more willing to accept a higher demand in a greater number of consolidated employment. The score calculator is an iterative component to measure the performance of the present VM. This method is also useful for implementing incentive based scheduling, which focuses on punishing (for example, exclusion) volatile, low capacity, selfish, or malicious resources. As a result, score-based scheduling motivates volunteers to offer their time and resources gladly and consistently. Let us say that VM_i has the following attributes:

$$A_{ti} = (A_{i1}, A_{i2}, ..., A_{in}),\ n = 1, 2, \ldots 5$$

where A_{i1}, A_{i2}, A_{i3}, A_{i4} and A_{i5} represents MIPS, PE, RAM, storage and bandwidth.

The machine prioritization phase assigns a priority to each virtual machine (VM) based on its computational and communication capabilities. Virtual machines that have been deployed in the cloud can be used to form a virtual cluster. Virtual machines come in a variety of shapes and sizes. This prioritization is based on each resource's processing speed, number of processing cores, RAM size, bandwidth strength, and storage size. The proposed scientific workflow scheduling algorithm uses a score system to select only those machines for scheduling that have the shortest total execution time from the server score center.

In most cases, computing a task node priority entails calculating the most cost-effective path from the node to the entry and exit nodes by adding up the node's computational and

communication costs along that path. Unfortunately, the completion time of one task's successors on different processing resources in different time slots varies due to heterogeneity and fluctuation in resource performance. As a result, determining the true urgency of this task is extremely difficult. Several performance measurements, such as the median or average value of resource performance, can be used to estimate the completion time of nodes in such a scenario. The average performance value is used to demonstrate this algorithm in the following discussion. To calculate a resource's average performance as precisely as possible, the scheduler must estimate the number of time slots required to complete the task. To do so, we proposed an algorithm for calculating each machine's score based on its performance. The following is a detailed description of how VM performance is calculated.

Algorithm 2: VM Parameter Collecting Algorithm

```
Input: VM_ID, USER_ID, VM_MIPS, VM_RAM, VM_STORAGE, VM_PE,
Output: VMCapacity: List  // List of the VM parameters and
Begin
     For each Datacenter in Cloud do
          For each VM in Datacenter do
               VMCapacity [ID] = VMCapacity [ID] +
               Datacenter.get(VM_MIPS, VM_RAM, VM_STORAGE,
               VM_PE, VM_BW)
          End For
     End For
     Return VMCapacity <list>
End
```

4.4.1.1 Scoring System

The cloud is a promising infrastructure for workflow execution because it provides a variety of computing resources and virtually infinite computing capacity. Cloud providers provide virtual machines to the general public, including scientists, through Infrastructure-as-a-Service, or IaaS. A cloud offers a wide range of computing resources and appears to be an appropriate infrastructure for running SWfs. However, because the estimation involves various parameters, such as VM types and different SWfs, the problem of choosing the number and type of VMs remains a critical problem for SWf execution in the cloud [4]. When Cloud Broker (CB) receives tasks to be scheduled, the Cloud Information Service (CIS) is queried to identify the services required to execute the tasks received from the user, and then the tasks are scheduled on the

discovered services. For example, in a given time interval, tasks T_1, T_2, T_3,... T_n could be submitted to CB. Because the processing elements (Virtual Machines) are heterogeneous in terms of processing speeds and memory, a task executed on different VMs will have a different execution cost. Assume that when CB receives the tasks, Virtual Machines VM_1, VM_2, VM_3,... VM_m are available. Our goal is to schedule tasks on virtual machines in order to achieve higher VM utilization with a shorter makespan.

The Scoring System is responsible for calculating the performance of the current virtual machine iteratively. The execution of scoring system begins with a search for available hosts and virtual machines in each hosts provided by the cloud provisioning system. For instance, during the retrieval of hosts the first host found can be named H_1 and Virtual machine in the host can be named H_1VM_1. List of these ready VM is given to the VM Score Calculator component. Then, the next process is fetching attributes of ready VM in the host and the corresponding values for each attributes. Host attribute extraction takes place iteratively based on the QoS parameters for every VM. Validity of the VM is checked and virtual machines that cannot be managed by the existing host capability is filtered out. Once the performance of each VM is calculated, then the attributes of each host are returned. Therefore, total host level score is calculated. Final score for each virtual machine is stored in Server score center in decreasing order.

4.4.1.2 Normalise Attribute Values

To rank the performance of VMs for an attribute group, the attribute values are normalised. The relative performance of VM in each group can be seen using the attributes group based rank. A VM that performs well in the memory attribute, for example, may perform poorly in the network bandwidth performance (Varghese et al., 2016). The normalised value of each attribute can be calculated by:

$$\bar{n}_{i,j} = \frac{n_{i,j} - \mu_j}{\sigma_j}$$

Eq. 12

Where μ_j – the mean value of attribute $n_{i,j}$ over m number of VMs, and σ_j is the standard deviation of the attribute $n_{i,j}$ over m VMs.

The mean value is the sum of all values of an attributes (MIPS, RAM, BW, Storage, PE) for *m* number of VMs and dividing that sum by *m*. Whereas, the standard deviation is the sum of the squares of the difference between the value of each attribute and mean for *m* number of VMs and dividing that sum by *m*. The standard deviation shows that the average amount of variability found in the data or dataset. It tells, on average, how far each value of data entry lies from the mean. A high standard deviation denotes that values are far away from the average, while a low standard deviation shows that values are grouped near the average. It can be calculated using a sample standard deviation formula, which is given below.

$$S = \sqrt{\frac{\sum(X - \bar{x})^2}{n - 1}}$$

Eq. 13

Where *S* is standard deviation, *X* is the value of each attributes of *VM*, $\bar{x}$ is the mean value and *n* is number of VM. The normalised attribute values are denoted as $\overline{N}_{i,k} = \{\bar{n}_{i,1}, \bar{n}_{i,2}, \bar{n}_{i,3}, \ldots\}$, where $i = 1, 2, \cdots m$, for *m* is number of VMs and $k = 1, 2, \cdots, p$, and *p* is the number of attributes. In our case, we have five attributes, namely: MIPS, RAM, BW, Storage and PE. So that, $p = 5$. Some attributes may be more important than others for a given application (Varghese et al., 2016). However, we chose not to use weights to prioritize attributes. The score of each VM is calculated as:

$$S_i = ((\overline{N}_{i,k})/p) = \frac{\sum_{i=1}^{p} \bar{n}_{i,k}}{p}$$

Eq. 14

Where *p* is the number of attributes in the attribute group. The scores are sorted in a decreasing sequence of score value, which is for generating the ranking of the VMs, RVM_i, based on performance i.e. from most powerful one to the least powerful one.

Algorithm 3: Virtual Machine Scoring Algorithm

```
Input: VMCapacityList: List
Output: VMScoreList: Collection
Begin
    For each Data center (DC) in the Cloud do
        For each Host in the DC do
            For each vm_i ∈ VM do
                Get values for VM_ID, VM_MIPS, VM_RAM, VM_BW,
                VM_Storage, VM_PE
                For i = 0;i < VM.size;i++
                    Normalise VMCapacityList.Values()
                    VMScore = ((VMCapacityList.Values())/5)
                    Score VM using S_i
                    Generate ranks of VM (RVM_i
                End for
            End for
            VMScoreList = VMScore.Sort()
        End for
    End
End
```

4.4.2 VM State Manager

The VM state manager is responsible for the summation of host level and DC level scores for the purpose of identifying the VM state. The VM state manager checks the status of every VM after VM scoring component completes evaluating the performance of machines. Sometimes a server may have a less than tolerable score. In such scenario, it should be removed and removal of VM can be done with the help of VM Ejector. Finally, the whole data gained from resource centers, hosts and virtual machines is translated into a lasting model. Threshold values of the VM is calculated form the resource utilization and the farthest distant from the mean values of VM capacity score. To calculate the farthest distant from the mean values of VM capacity score, we used the variation between the normalized attributes of each VM and the mean of attribute values which is given below:

$$Threshold_{min} = \min\left(U_i \sum_{i=1}^{m}(n_{i,j} - \bar{x})\right)$$

Eq. 15

Where U_i is the utilization of VM i, $n_{i,j}$ is the normalised j^{th} attribute of i^{th} VM, and $\bar{x}$ is the mean attributes for *m* number of VMs. For a resource r_i, we can define the utilization U_i as (Gourisaria et al., 2018):

$$U_i = \sum_{j=1}^{n} u_{i,j}$$

Eq. 16

Where *n* is the number of tasks running at the current time, and $u_{i,j}$ is the usage of resource of a task t_j.

4.4.3 VM Ejector

Given the status of the VM from VM state manager, this component can discard VM form the list. This component accepts the VM status as an input and isolates the VM that should be removed. In case there is a VM with very low capacity or malicious, VM ejector removes these functions from the resource lists ready for the next scheduling.

Algorithm 4: Virtual Machine Ejecting Algorithm

```
Input: VMScoreList: Collection //Sorted List
Output: EjectedList: List // List of VM with very poorest
Begin
    For i = 0;i < VMScoreList.size;i++
        If Score[i]< Threshold_min then
            EjectedList.Add(VMScoreList[i])
            VMScoreList[i].Remove()
        End if
    End  for
End
```

4.4.4 Task Prioritizer

It is a tough problem to identify the critical tasks to be scheduled at each step to certain VM in order to schedule a task graph efficiently. The reason for prolongation of the schedule length may be due to the delay of critical tasks. When we talk about task priority, we're talking about how important it is to think about the task. A task priority factor is used to determine the importance of a task. A task with a high priority represents a high level of concern and should be processed first; on the other hand, a task with a low priority represents a low level of concern and should be processed after higher priority tasks have been completed. A task node's preference is usually determined by calculating the maximum distance i.e. computational and communication costs between it and the entry and exit nodes. A task graph's critical path (CP) is a set of nodes and edges that form a path from an entry node to an exit node, all of which have the same maximum rank value.

The task prioritization phase determines the priority of tasks and assigns it to each task in the task graph. A task's priority is calculated using both upward and downward rank, which are based on two attributes. Attributes we used to compute priority includes Average computational cost and Data transferring cost. When a task node is scheduled, all of its successors' ranks are updated. The scheduler selects the unscheduled task with the highest final rank value from the task ready queue (TRQ) at each scheduling step. When all of a task's predecessors have been scheduled, it is placed in TRQ. The average performance of processing nodes in feasible time slots is used to update the rank of a task node.

Task computational cost is the average cost of computation across all q processors, as shown on the task graph. The formula for Average Computational Cost (ACC) is given in the previous chapter in eq. 3. The cost of data transfer for task t_i is the cost of communication or the time it takes to transfer data from task t_i to all of its immediate successors [3]. The cost of data transfer is calculated at every level and given as follows:

$$\mathrm{DTC}\ (\mathrm{t_i}) = \begin{cases} \sum_{j=1}^{n} C_{i,j}, & i < j \\ 0, & for\ exit\ node \end{cases}$$

Eq. 17

Where n – the number of nodes in the next level in DAG, $C_{i,j}$ – the average communication cost between node i and its successors like node j.

For exit tasks, the DTC is zero, since it has nowhere to communicate the data. Average communication cost ($C_{i,j}$) between node i and node j is given by eq. 3. Those algorithms prioritize tasks to be scheduled by assigning a weight to each node typically, the corresponding computation cost and edge which is the corresponding communication cost based on a value computed by a rank function; this is usually a function of the weights assigned to the nodes and edges of the graph. Although it has long been known that the rank function (and the values it returns) can affect the quality of the schedule produced in homogeneous environments, we considered both upward and downward ranking when prioritizing tasks. Each task has its priorities with these upward and downward ranks values, based on its topological order of tasks in the workflow structure (DAG), and the preceding constraint is preserved. Downward rank is the longest distance from the entry task to task. Similarly, the downward rank of tasks can be computed recursively by traversing the task graph downward starting from the entry task the DAG. The downward rank for the entry task is zero. Downward rank is calculated by:

$$rank_d(t_i) = max_{t_j \in pred(t_i)}\{rank_d(t_i) + \bar{w}_j + \bar{C}_{j,i}\}$$

Eq. 18

Where $\boldsymbol{pred}(\boldsymbol{t_i})$ - the set of immediate predecessor of the task t_i, $\boldsymbol{\bar{w}_j}$ - average computational cost of task t_i, $\boldsymbol{\bar{C}_{j,i}}$ - the average communication cost of edge between task t_j and t_i.

The final rank is computed for each task based on its *ACC*, *DTC*, $rank_u$ and $rank_d$. The final rank of the task is given by:

$$rank_f = round\{ACC + DTC + rank_u + rank_d\}$$

Eq. 19

Algorithm 5: Task Ranking Algorithm

```
Input: TaskList from Job Queue
Output: SortedTaskList
Begin
   For each task t_i in DAG do
      Calculate the average computation cost (ACC), data
      transferring cost (DTC), upward rank (rank_u) and
      downward rank (rank_d)
      If task t_i is entry task then
        rank_f(t_i) = rank_u + DTC
      End if
      Else if task t_i is exit task then
         rank_f(t_i) =  rank_d + ACC
      End if
      Else
          rank_f(t_i) =  ACC + DTC + rank_u + rank_d
      End
   SortedTaskList = rank_f(t_i).Sort()
   End for
End
```

4.4.5 VM-Task Mapper

High priority tasks are selected from the TRQ and assigned to the right machine at this phase. The highest score value is the best processor. Although it attempts at arranging tasks on or near the node where the data are situated to minimize data transmission overheads. After the entry task is mapped to VM, from the immediate successor tasks it selects the task with the highest DTC to map it onto the same machine as the predecessor task. So that, the task which is data-intensive is selectively assigned to the same VM as its parent, which could minimize data transfer cost to some extent.

Algorithm 6: VM-Task Mapping Algorithm

```
Input: SortedTaskList, VMScoreList
Output: VM – Task Mapping activation
Begin
   While there are unscheduled tasks in the SortedTaskList do
      If t(i) is entry task then
        Select VM j from the top of VMScoreList
        Assign t(i) to VMj
      End if
      Else If t(i) is mapped to VMj and t1, t2, t3,…tn is a set
      of immediate successors task of t(i) then
        Compare DTC of succ(t(i))
        Select the task with highest DTC from the succ(t(i))
        Select VMj from the VMScoreList
        Assign succ(t(i)) to VMj
      End else if
   End while
End
```

CHAPTER FIVE

EXPERIMENTAL SETUP AND IMPLEMENTATION

Introduction

This section discusses the experimentation environment, simulation tools and methodologies used to perform the experiment that have been proposed in this research paper. Our evaluation is based on a series of comprehensive evaluation metrics. In this section, we discuss the evaluation metrics first, and then shows the evaluation results in both simulation workload and real-world workloads.

5.1 Simulation Tools Used in the Research

5.1.1 Hardware Requirement

- During implementation of our proposed system, we used the following hardware:
- Processor – Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz 2.60 GHz
- Installed RAM – 8.00 GB (7.89 GB usable)
- System Type – 64-bit operating system, x64-based processor
- Hard Disk Size – 1TB

5.1.2 Software Requirement

- To implement SBHEFT, we have used the following software application:
- Operating System – Windows 10 Home
- Integrated development environment (IDE)
- Framework – Apache Netbeans (12.1 release)
- Programming Language (Platform) – Java
    - Java™ Platform Standard Edition 16 Development Kit - JDK™ 16
    - Java version "1.8.0_281"
    - Java(TM) SE Runtime Environment (build 1.8.0_281-b09)
- Simulation Tool – WorkflowSim 1.0

5.1.3 Java Development Kit

The Java Development Kit (JDK) is a software development environment that includes a set of tools and libraries for creating Java programs. The JDK is required to convert your source code

into a format that can be executed by the Java Runtime Environment (JRE). The JDK includes tools useful for developing, testing, and monitoring programs written in the Java programming language and running on the Java platform. The Java Runtime Environment (JRE), a Java interpreter/loader (java), a Java compiler (javac), a Java archiver (jar), a Java documentation generator (javadoc), and other Java development tools are all included (Kostaras et al., 2020).

The NetBeans Platform is a large Java framework on which large desktop applications can be built. The NetBeans IDE is one of the hundreds of NetBeans Platform-based applications. The NetBeans Platform includes APIs that make it easier to work with windows, actions, files, and a variety of other things that are common in applications (Kostaras et al., 2020). To simulate our work, Apache NetBeans integrated with WorkflowSim 1.0 is used. Apache NetBeans is a full IDE for Java SE, Java EE, PHP, JavaScript, HTML5 and more, including some support for Groovy and (new) C/C++.

5.2 Experimental Setup

In this section, we provide an evaluation of the performance of our algorithm in relation to certain performance metrics, appropriate workloads and the simulation environment. The environment we are thinking of is a heterogeneous computer resource linked to heterogeneous connections. The platform is represented in a configuration file for our simulation, with the following information: the number of tasks that compose each application and the platform's numbers of computational resources.

5.2.1 WorkflowSim

For the experimentation of our work, we used a WorkflowSim 1.0 which is an extended simulation toolkit from CloudSim for workflow simulation. Weiwei Chen, a Ph.D. student from University of Southern California, developed WorkflowSim under the Apache License version 2.0. Many static and dynamic scheduling policies are supported by WorkflowSim including those that respect parent–child relationships between tasks and jobs in original or clustered workflows. Some of algorithms currently supported by WorkflowSim are described below (K. Kumar & Thaman, 2018).

MinMin - On a VM with remaining capacity, the Min-Min scheduling algorithm schedules the next cloudlet with the shortest length. As a result, smaller tasks are scheduled on slower machines.

MaxMin - The most appealing scheduling algorithm has been the max-min scheduling algorithm. It works on the principle of scheduling cloudlets with the longest length to VMs with the smallest remaining capacity. It reduces the execution time of the smallest cloudlet by exposing it to the fastest VM.

RR - Cloudlets are scheduled on idle VMs in ascending order of their Id using the RR scheduling algorithm. This is the most basic type of scheduling, and performance is determined by the amount of allocation. FCFS is equivalent to large quantum size. The RR algorithm is the default preemptive algorithm. RR is considered non-preemptive in WorkflowSim.

5.2.2 Experimentation

We repeatedly performed the experiment 10 times and the average of the recorded makespan value is used to arrive at the results for each algorithm. In all experiments, we used 2 DCs and four sets of VM size or node size; 5, 10, 15, 30. In our experiments, we have tried to simulate different size of hosts and data center as well. We used a host in a single data center, which comprises single host, and a single data center with different size of hosts. Here is the trick to use multiple data centers in one broker in WorkflowSim simulator. Broker will first allocate all VMs to the first datacenter and if there is no enough resource then it will allocate the failed VMs to the next available datacenter. The trick is make sure your datacenter is not very big so that the broker will distribute them.

Entity Name	Parameter	Value	Comment
Host	Processing Speed (MIPS)	2000	Default value in WorkflowSim
	RAM	2048	Default value in WorkflowSim
	Bandwidth (Mb)	10000	Default value in WorkflowSim
	Storage Size (GB)	1000000	Default value in WorkflowSim
	PEs Number (#)	2	Default value in WorkflowSim
VM	Image Size (MB)	10,000	Default value in WorkflowSim
	RAM Size (MB)	Random value	We used random values between 1 and 1000
	Bandwidth (Mb)	Random value	We used random values between 100 and 10000

	Processing Speed (MIPS)	Random value	We used random values between 100 and 2000
	PEs Number (#)	1/2	We used 1 or 2 for PEs number
	Storage Size (GB)	10,000	We used random values between 1000 and 1000000
Data Center	Architecture	x64	Default architecture in WorkflowSim
	OS	Linux	Default OS in WorkflowSim
	VM Name	Xen	Default name in WorkflowSim
	VM Policy	Time-Shared/ Space-Shared	WorkflowSim supports both scheduling policy.
	Time zone this resource located	10.0	Default value in WorkflowSim

Table 3 DC, Host and VM parameters used in the simulation

The test was performed with different random values generated with *Random(System.currentTimeMillis())* of java method for MIPS, PEs, Storage, Bandwidth, and RAM values. For our measurements, we assigned random values for VMs performance metrics we used since we used a single system, not distributed system. For core numbers we used the default values (either 1 or 2) on the WorkflowSim tool running on the basis of a data center with the fully connected network topology. Furthermore, we assume that the transfer between two VMs is constant and not equal to zero. The following parameters for the VM, host and the data center have been used during the whole experiment.

5.2.3 Workflow Applications

In order to evaluate the proposed system, we considered synthetic workflows with different types of artificially-generated properties only. Even though we tried to validate our work on public clouds with the real-world workflow applications, we could not afford it and we will retry it in the near future in any possible case. For this case we only tested our work on simulation environment of WorkflowSim with synthetic workflow applications. The workload task is assumed to be scheduled on a virtual heterogeneous platform with multiple virtual processors and different processing frequencies for each processor. We chose four different synthetic scientific workflows published by Pegasus project; CyberShake, Inspiral (LIGO), Epigenomics and Montage workflow. These applications have been chosen because they involve a wide range

of scientific disciplines and resource requirements. Two of them are data-intensive (Montage and CyberShake) and the other two workflows are CPU-intensive (Epigenomics and Inspiral) applications. As per Juve et al. (2013) characterization, Epigenomics is CPU-bound workflow, Montage is I/O bound workflow, LIGO Inspiral and CyberShake are described as a workflow with both large memory requirements and large resource requirements.

The workflow of CyberShake is a data-intensive application par excellence. It is a seismology application used to characterize earthquakes through the production of synthetic seismograms. Montage, is another astronomical data-intensive workflow, using a collection of input pictures it is used to create unique sky mosaics. The LIGO Inspiral is an application for the sensing of gravitational waves in the domain of physics. This program is deemed to be CPU-intensive (Bharathi et al., 2008; Juve et al., 2013).

We used different size of cloud applications as a set of tasks. This number of tasks represents the workload of workflow applications. To evaluate the performance of the proposed algorithm, four sizes of workflow instances are used; small (25, 30 tasks), medium (46, 50 tasks), large (100 tasks) and extra-large (997, 1000 tasks). A series of experiments have been conducted to evaluate our work and compare it with numerous existing state-of-the-art cloud scheduling algorithms. The following table shows the basic structures and types of the four workflows used in our experiments.

Workflow	Resource Type			Number of Tasks	Number of Cloud Instances
	Memory	***CPU***	***I/O***		
CyberShake	Low	Medium	Low	30, 50, 100, 1000	5, 10, 15, 30
Epigenomics	Medium	High	Low	24, 46, 100, 997	5, 10, 15, 30
Inspiral (LIGO)	Medium	Medium	Low	30, 50, 100, 1000	5, 10, 15, 30
Montage	Low	Low	High	25, 50, 100, 1000	5, 10, 15, 30

Table 4 Types and description of Workflows used in the evaluation

5.3 Evaluation Metrics

In this section, we discuss the evaluation metrics first, and then shows the evaluation results in simulation workload. Our study uses synthetic workflow definitions of varying sizes of up to

1,000 tasks that have been widely used as benchmarks in a variety of projects. We compare the performance of the proposed algorithm with three algorithms based on the performance metrics schedule length.

5.3.1 Makespan

The makespan or schedule length, which defines the actual finishing time of the last task in a DAG, is a popular metric in task scheduling. The makespan is the total amount of time it takes to complete all of the jobs that have been received. It is the overall completion time of the exit task after all tasks in the graph have been scheduled. That means the makespan is equal to the maximum actual Finish Time (FT) of the exit node in the DAG. It is the maximum completion time or the time it takes for the IaaS Cloud system to complete the most latest job. From the perspective of execution time, this parameter indicates the quality of job assignment to resources. The shorter the makespan, the more efficient the algorithm, which means it takes less time to perform the algorithm. Therefore, makespan is given by eq. 9:

$$makespan = \max\{FT(t_{exit})\}$$

5.3.2 Performance Improvement Rate Percentage (PIR (%))

The PIR is calculated using the following equation eq. 18 and is defined as the percentage of improved performance or drop in makespan for the proposed approach (let us say *i*), with comparison the other algorithms (let us say *j*) (Abdulhamid et al., 2016).

$$PIR(\%) = makespan(j) - makespan\ (i) * \frac{100}{makespan\ (i)}$$

Eq. 20

CHAPTER SIX

RESULT AND DISCUSSION

Overview

In this section, we discuss the result of the experiments that had been carried out in order to evaluate the proposed approach, the proposed algorithm performance. To simulate a cloud with several data center platforms, we used a simulator based on the WorkflowSim simulation tools. We compared our system with workflows from several scientific disciplines with different structures using a standard technique. The purpose of the system evaluation is to see how effective is the performance of the system in comparison with other pre-existing algorithms. It is worth mentioning that all schedules used in this section for experiments.

6.1 Results

First, our experiments aim to study the performance of our approach while scheduling scientific workflows with increasing complexity. For this purpose, we varied the size of the workflows in a number of tasks. Our proposed algorithm enhances HEFT through selection of resources (VMs) and execution order of task, which reduces makespan and increases the reliability of the selection of resources. We compared the proposed algorithm with Round Robin, MaxMin and MinMin scheduling algorithms. These scheduling algorithms are widely used on cloud computing environment. A Round Robin mechanism has been developed to evenly split schedule time amongst all scheduled tasks. It includes all tasks in the waiting list and assigns a little unit of time for each activity. When the period for the first task has expired, the programmer selects a task and assigns it to the controller. Quantum time is the interval of time between processes. This is the cyclical technique, in which the controller at least once assigns all jobs and the programmer takes over the first task. MaxMin and MinMin are the heuristic approaches used to solve the problem of task scheduling in Cloud computing. The MINMIN heuristic selects the earliest minimum task from all available tasks and assigns it to a VM capable of presenting the job's minimum completion time. It improves the whole job completion time and hence the makespan time.

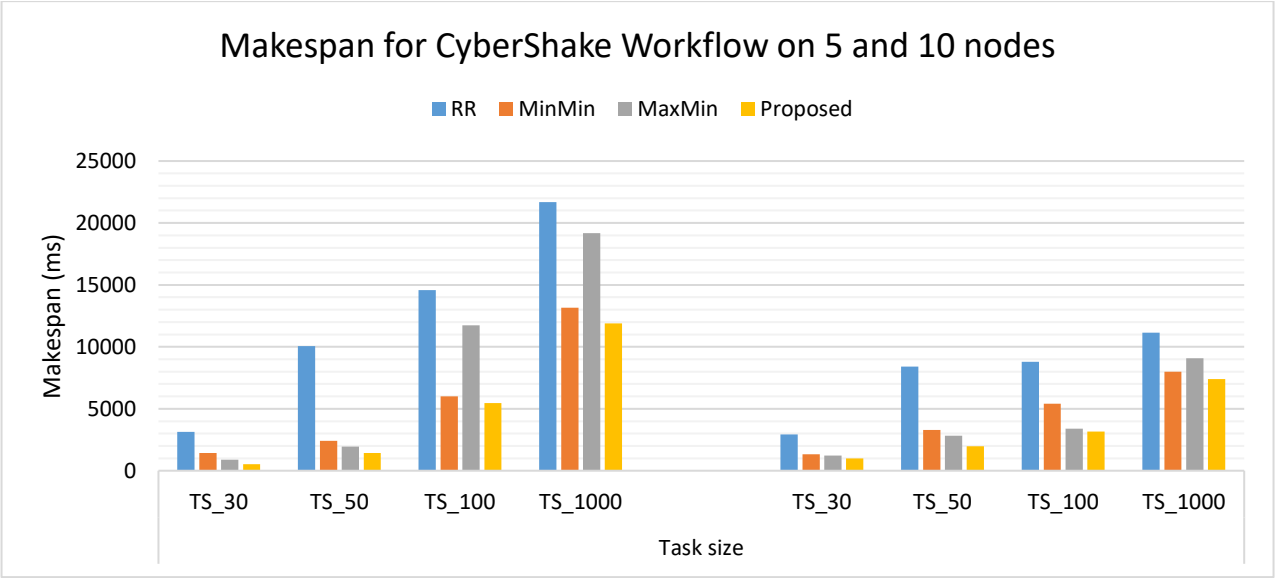


Figure 6. 1 Makespan for CyberShake Workflow on 5& 10 VMs

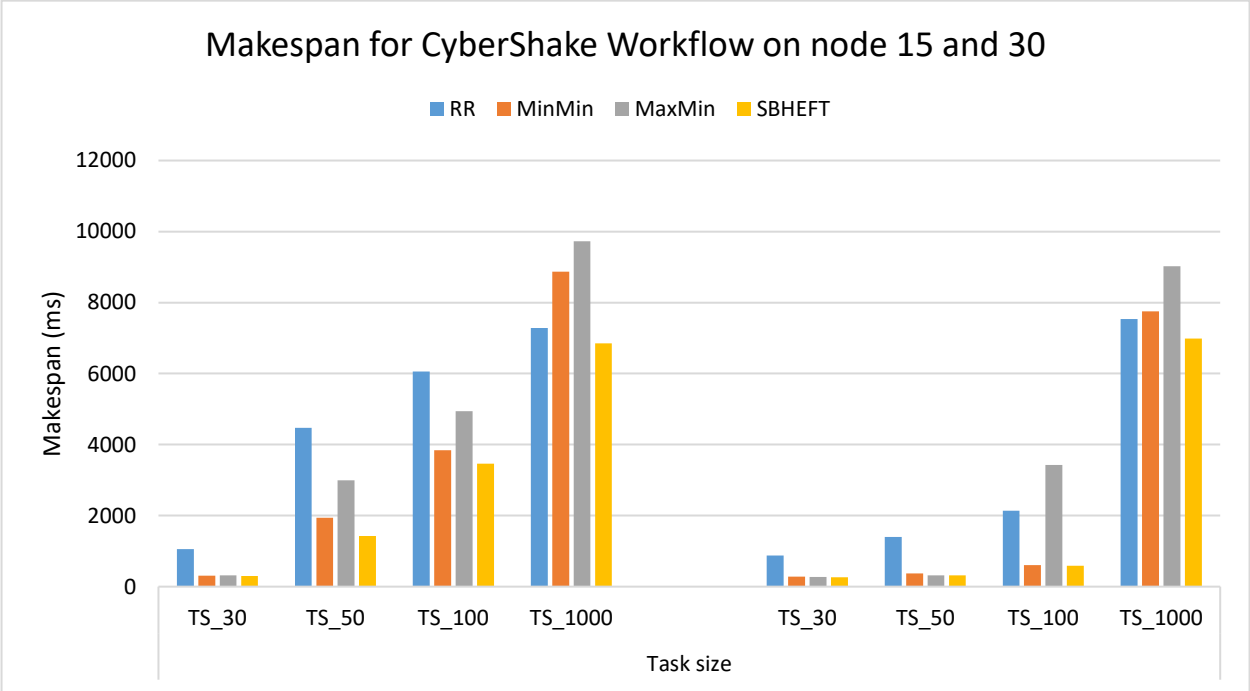


Figure 6. 2 Makespan for CyberShake Workflow on 15 & 30 VMs

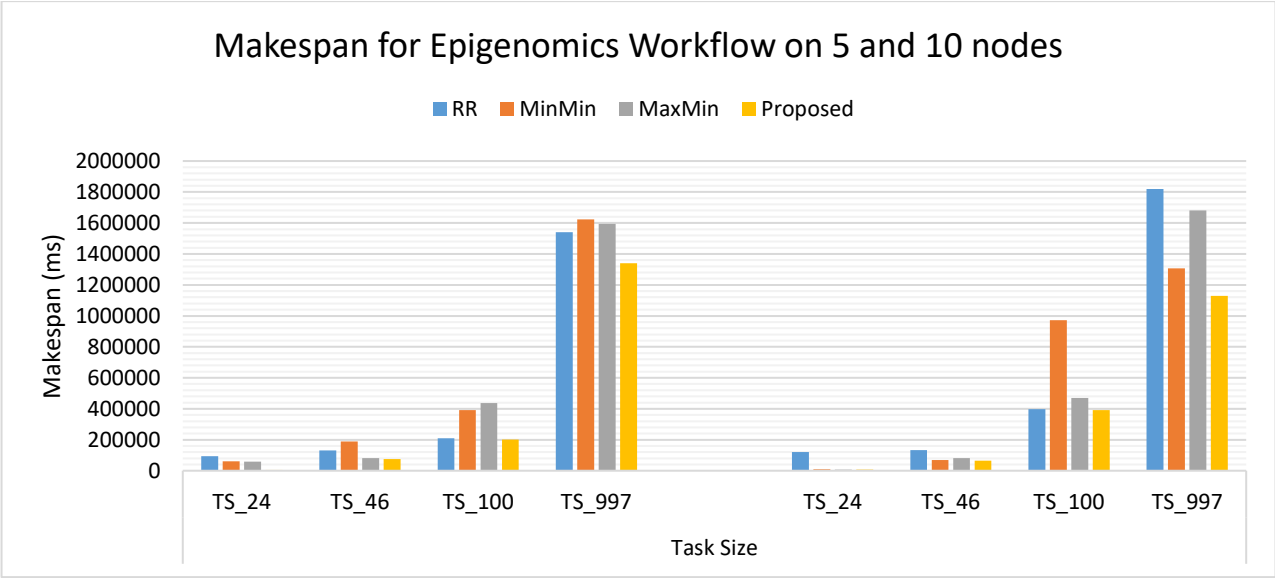


Figure 6. 3 Makespan for Epigenomics Workflow on 5 & 10 VMs

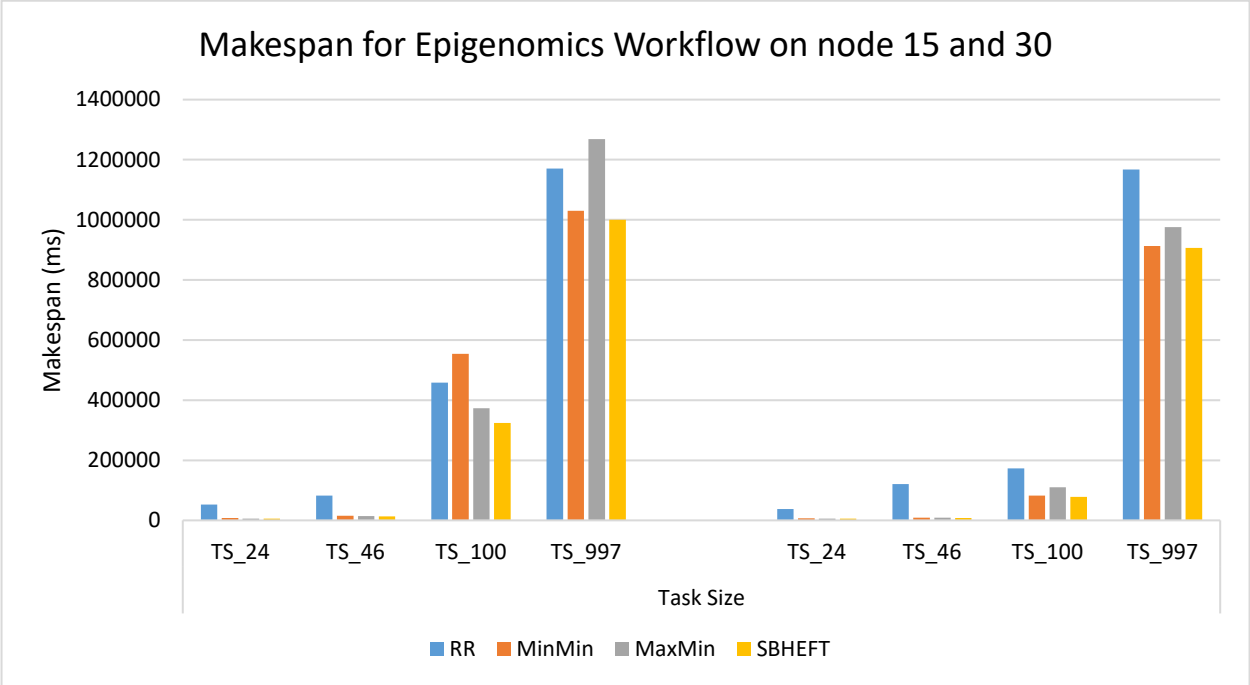


Figure 6. 4 Makespan for Epigenomics Workflow on 15 & 30 VMs

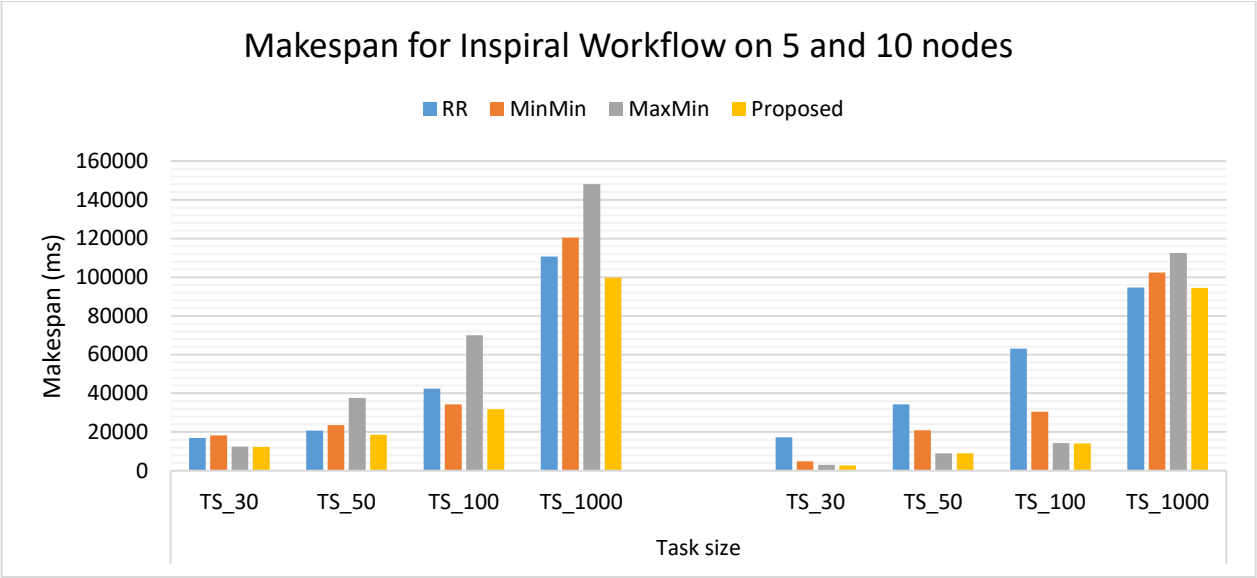


Figure 6. 5 Makespan for Inspiral Workflow on 5 & 10 VMs

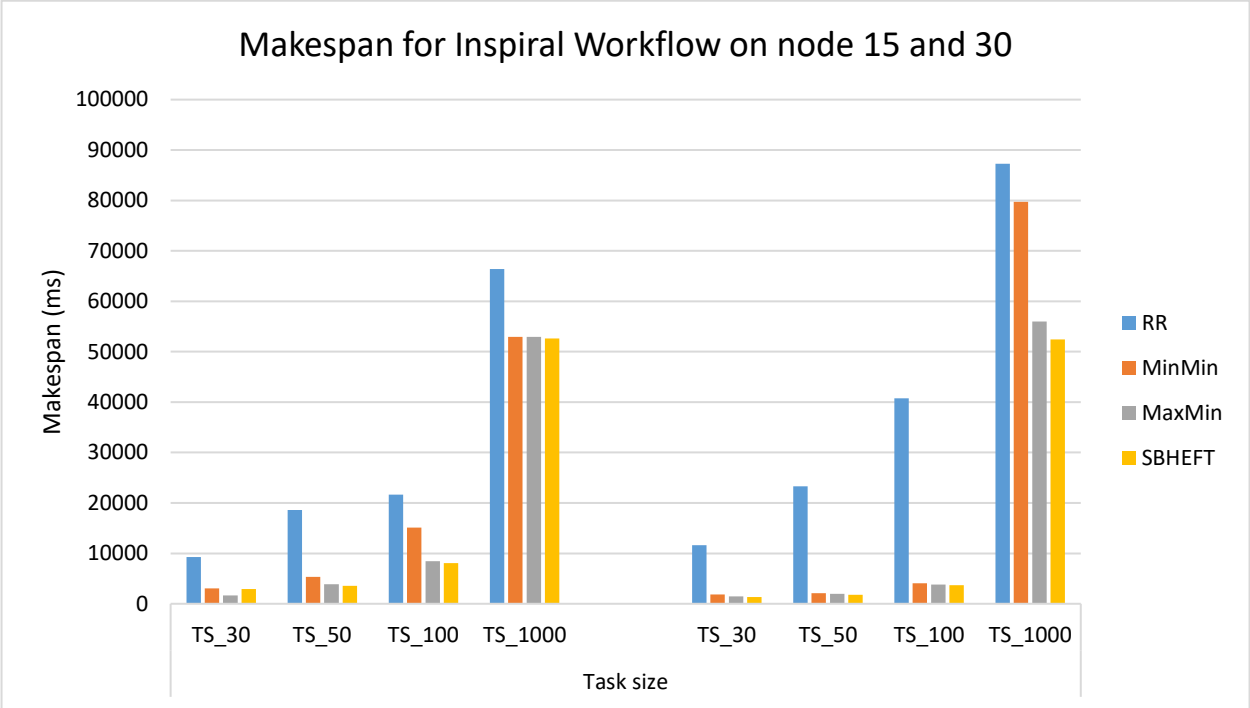


Figure 6. 6 Makespan for Inspiral Workflow on 15 & 30 VMs

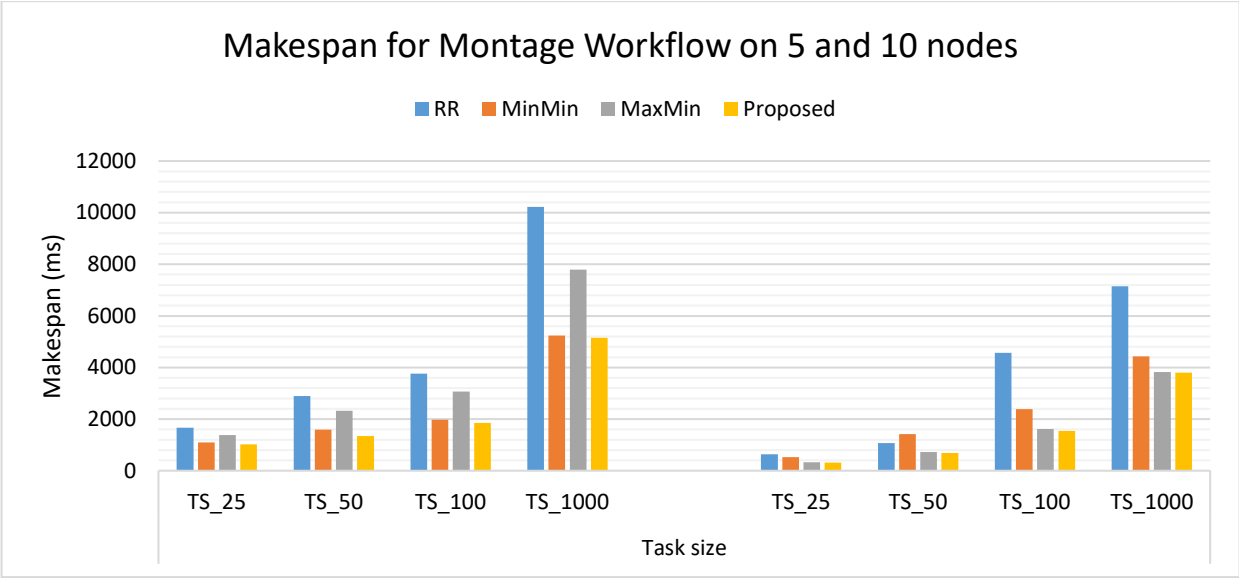


Figure 6. 7 Makespan for Montage Workflow on 5 and 10 VMs

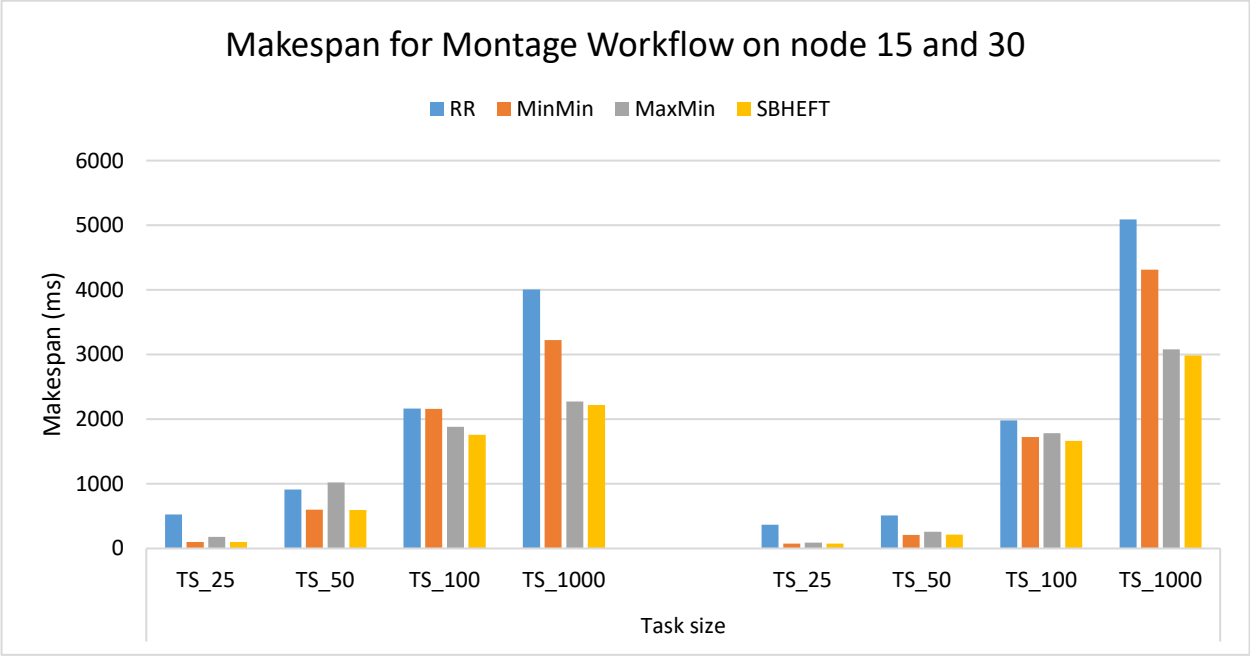


Figure 6. 8 Makespan for Montage Workflow on 15 and 30 VMs

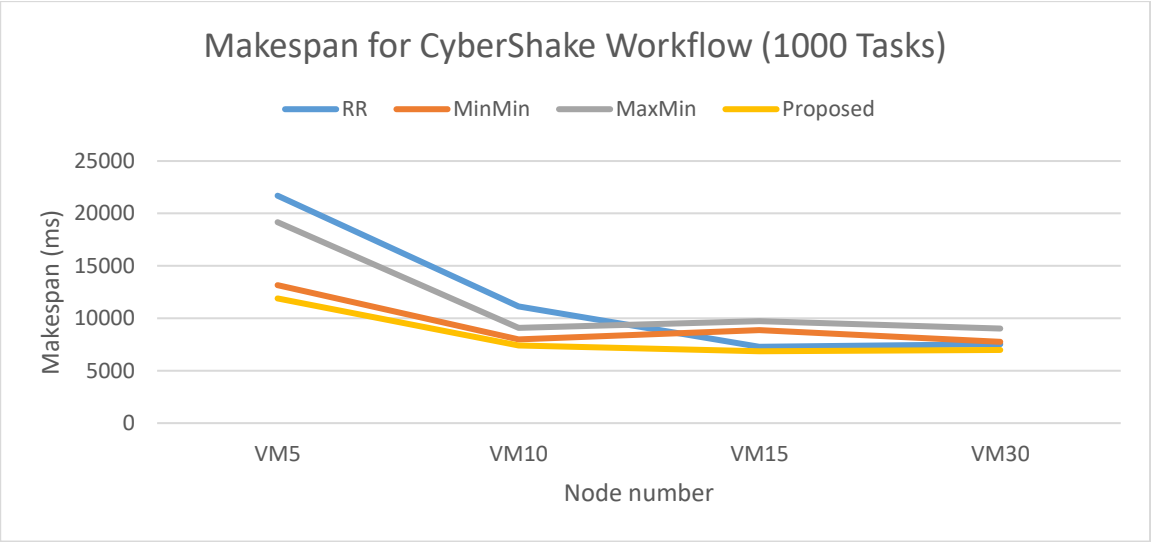


Figure 6. 9 Makespan for CyberShake workflow with 1000 tasks

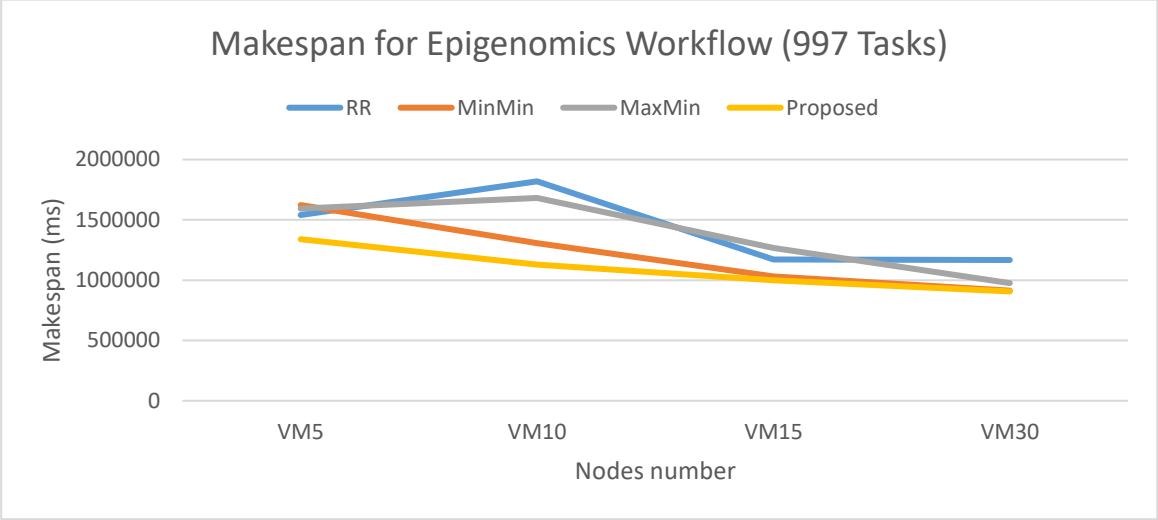


Figure 6. 10 Makespan for Epigenomics workflow with 997 tasks

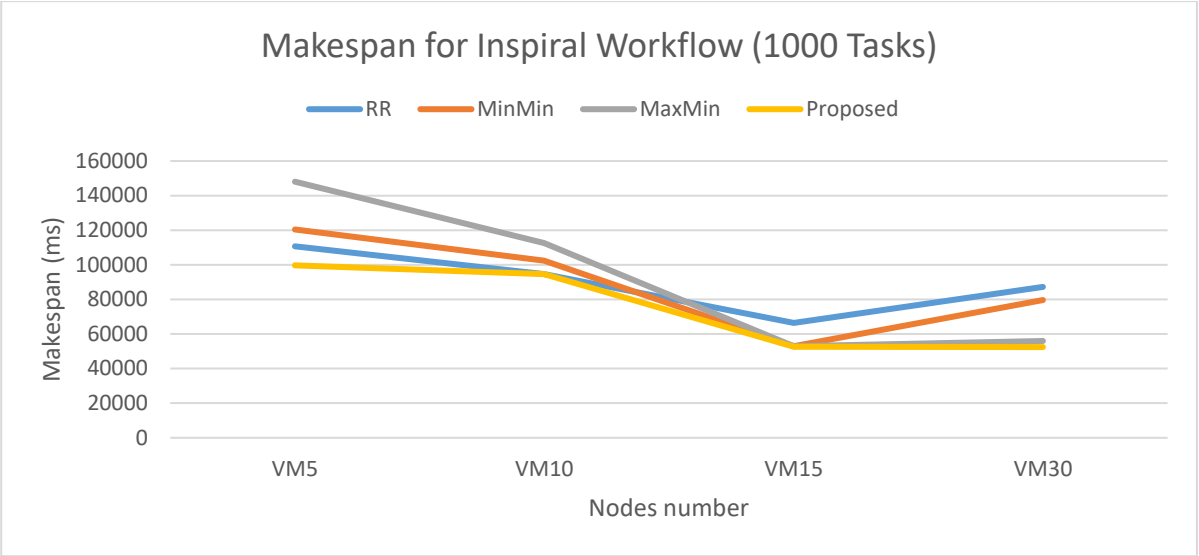


Figure 6. 11 Makespan for Inspiral workflow with 1000 tasks

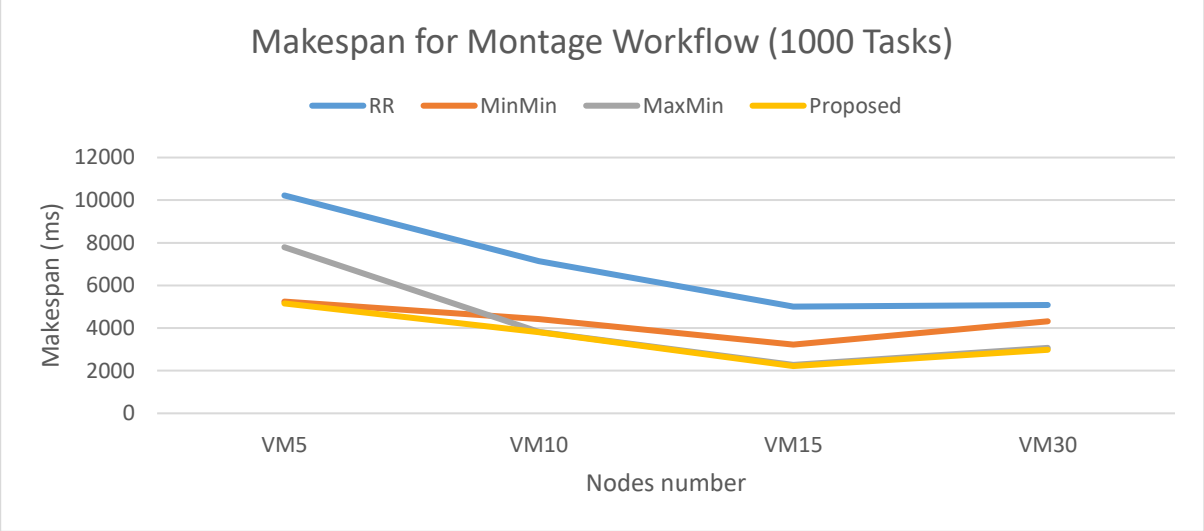


Figure 6. 12 Makespan for Montage workflow with 1000 tasks

6.2 Discussions

The experimental results that are plotted in figure 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7 and 6.8 represent makespan for different number of scientific workflows. The makespan of RR, MinMin, MaxMin and the proposed algorithm increases as the number of tasks in each workflow increases except in some cases, but is clearly improved by the proposed algorithm. All algorithm performs differently for different workflows. However, the result from all cases of the experiment suggests that, makespan of the proposed algorithm is lesser than the other three algorithms even

as the task size of the scientific workflows increase. Fig 6.1 shows the difference in makespan produced by RR, MaxMin, MinMin and the proposed algorithm for different task size of CyberShake workflow on 5 and 10 VMs. In CyberShake workflow experiment, RR showed the worst result of all algorithms. When compared to RR, our proposed scheduling algorithm provided makespan values that were roughly 83.08% lower for CyberShake with 30 tasks on 5 VMs. Consequently, the proposed algorithm decreases the makespan of CyberShake workflow with 30 tasks by 63.27% and 41.13% for MinMin and MaxMin algorithms on 5 VMs respectively. Comparing our work to RR, the makespan is decreased by 85.84% for Cybershake workflow of 50 tasks on 5 nodes. In addition, for MinMin and MaxMin, 40.82% and 26.82% respectively.

For CyberShake workflow of 30 tasks, the proposed algorithm makes lower makespan of 71.93%, 3.56% and 6.44% for RR, MinMin and MaxMin on 15 nodes respectively. For the same workflow, it improves the makespan by 70.09%, 8.5% and 4.38% for RR, MinMin and MaxMin on 30 VMs respectively. Therefore, we can say that, our proposed algorithm demonstrated the improved makespan for CyberShake workflow than any the above algorithms.

As shown in fig 6.3, for Epigenomics workflow, almost all algorithms performed well with consistent performance. However, the proposed algorithm still outperformed all the other three algorithms. For instance, for Epigenomics workflow with 46 tasks, the proposed algorithm improved the makespan by 42.94%, 60.31% and 7.58% when compared to RR, MinMin and MaxMin respectively, on 5 VMs. For the same workflow with 997 task size, it decreases the makespan by 13.05%, 17.44% and 15.98% compared to RR, MinMin and MaxMin respectively. For Epigenomics with 100 tasks on 15 computing nodes, the proposed algorithm showed good result by improving the makespan by 29.34%, 41.56% and 13.06% respectively for RR, MinMin and MaxMin. As a result, the proposed algorithm outperformed RR, MinMin and MaxMin in Epigenomics workflow.

For Inspiral workflow, as illustrated in fig 6.5, MaxMin showed the worst result for experimentation on 5 and 10 computing nodes. As the size of task increased but, RR performed badly. In all scenarios, the proposed algorithm surpassed in the performance than all other algorithms in the experimentation. The proposed algorithm showed the result that lessen the makespan by 27.45% on computing nodes compared to RR. For MinMin and MaxMin, 33.24%

and 1.76% respectively on 5 nodes. On the 15 computing nodes, the proposed algorithm decreased the makespan of the Inspiral workflow with 1000 tasks by 20.77%, 0.64% and 0.62% compared to RR, MinMin and MaxMin respectively. On 10 computing nodes, for Inspiral workflow with 1000 tasks, the same algorithm lowers makespan about 0.08%, 7.67% and 15.99% respectively. In conclusion, all algorithms showed a good result, even though the proposed algorithm is the best of all.

Fig 6.7 compares makespan between RR, MinMin, MaxMin and the proposed algorithm for Montage workflow with different task size. The number of tasks is represented on the horizontal axis, while the makespan is represented on the vertical axis. The simulation results clearly show that the proposed method outperforms other heuristic algorithms in terms of makespan. Except for Montage, which has 1000 tasks for which MinMin outperforms the proposed algorithm by 0.15%, the makespan improvements by the proposed algorithm are highly noticeable for other methods as shown in fig 6.12. Because of the parallel structure of these workflows, numerous tasks in the same level with relatively minimum task execution time assigned to different computing nodes with MinMin. This is not the case with the proposed algorithm, which allows the execution order of parallel tasks based on their priority of communication and computation cost, which increased the makespan in this case. As is illustrated in the figure, RR showed the worst result of all.

For Montage of 25 task size, the proposed approach resulted in lower makespan about 38.87%, 7.09% and 25.89% when compared to RR, MinMin and MaxMin respectively on 5 computing nodes. On the same computing nodes, the proposed algorithm decreases the makespan by 50.93%, 6.54% and 39.7% compared to RR, MinMin and MaxMin respectively for M. Compared to RR, MinMin and MaxMin, the proposed algorithm showed lesser makespan of 18.86%, 18.7% and 6.73% for Montage workflow of 100 tasks on 15 computing nodes. Therefore, from the above result the proposed algorithm showed a good result compared to other heuristics in the experimentation.

The simulation result clearly reveals that the makespan produced by some algorithms fluctuated, i.e. increased or decreased as the task size of the workflow is increased. In fig 6.9, it is shown that the makespan of CyberShake workflow with 1000 tasks for all algorithms used in the experiment. Even though it was high in the beginning for fewer computing nodes, the makespan

is reduced as the number of VM is doubled. As the number of VMs increased, the makespan of all algorithms become less and less. For all other workflow, the same thing happened. Despite the fact that, the makespan of other algorithms fluctuates, the proposed algorithm performed well from the beginning, i.e. with fewer VMs and when the number of computing nodes increased. However, for Montage workflow, the makespan of all algorithms including the proposed algorithm is increased because of the numerous parallel structure of tasks in these workflows.

The percentage of improved performance or drop in makespan of the proposed algorithm is another performance metrics we used to show the relative improvement percentage of the proposed system in comparison to other scheduling algorithms. From the following figures, one can identify that the proposed algorithm performance improvement percentage increases as the number of tasks of the scientific workflow increases. The same way, whenever the number of VMs increases, PIR(%) of the proposed system also increases, when compared to RR, MinMin and MaxMin.

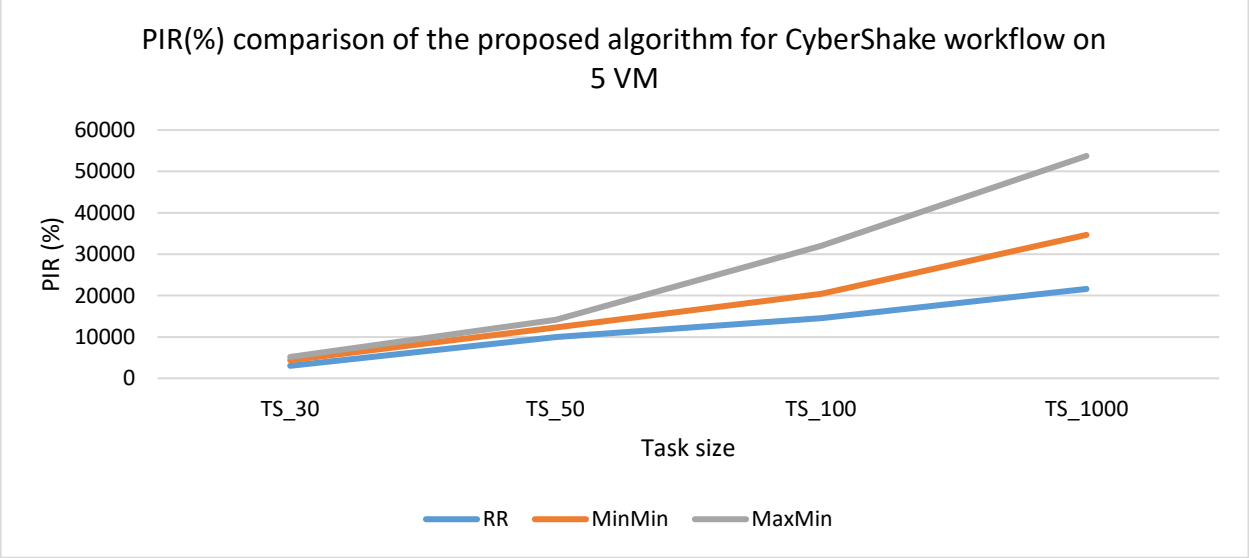


Figure 6. 13 PIR(%) comparison of the proposed algorithm for CyberShake workflow on 5 VM

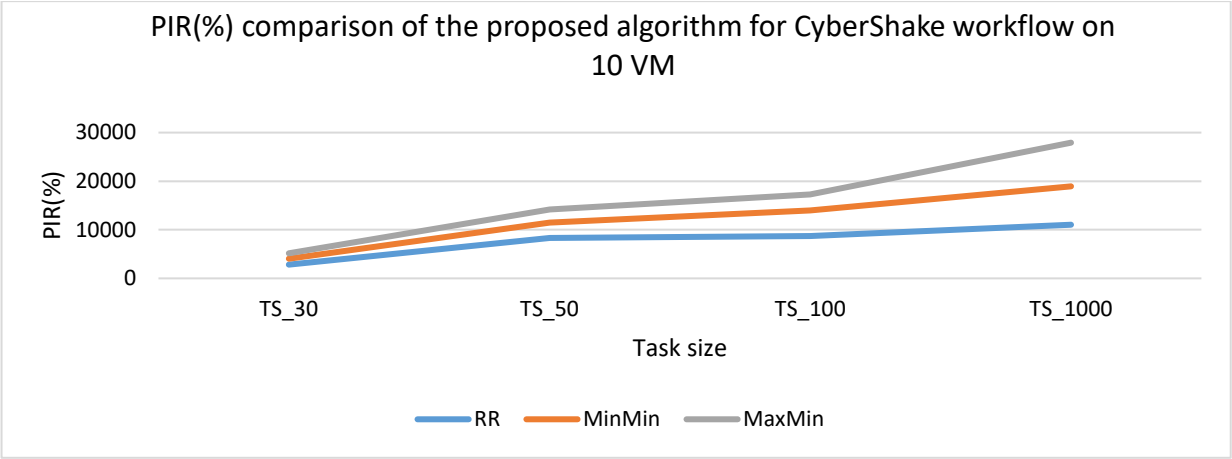


Figure 6. 14 PIR(%) comparison of the proposed algorithm for CyberShake workflow on 10 VM

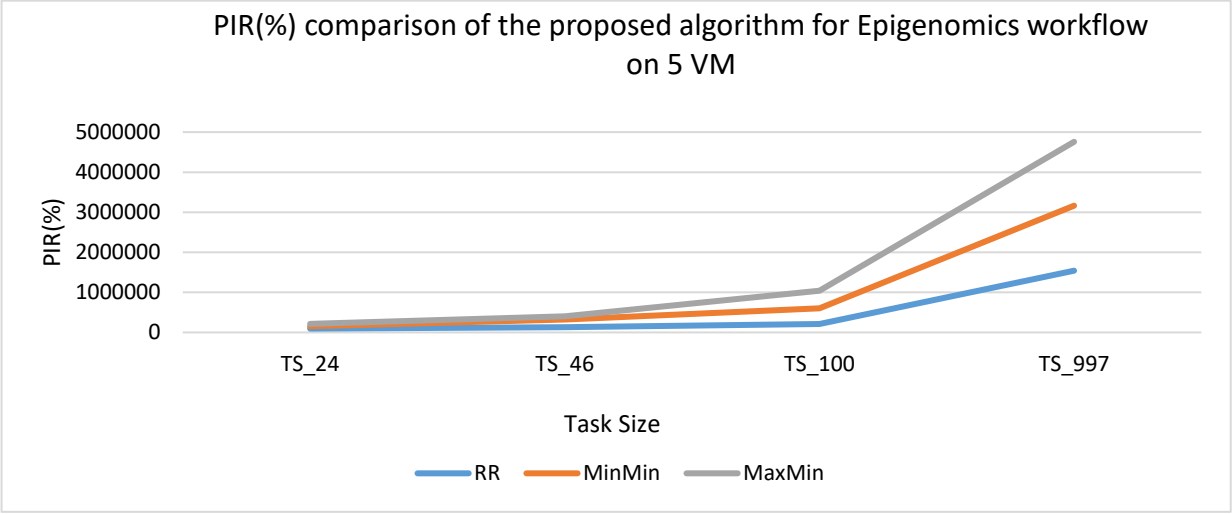


Figure 6. 15 PIR(%) comparison of the proposed algorithm for Epigenomics workflow on 5 VM

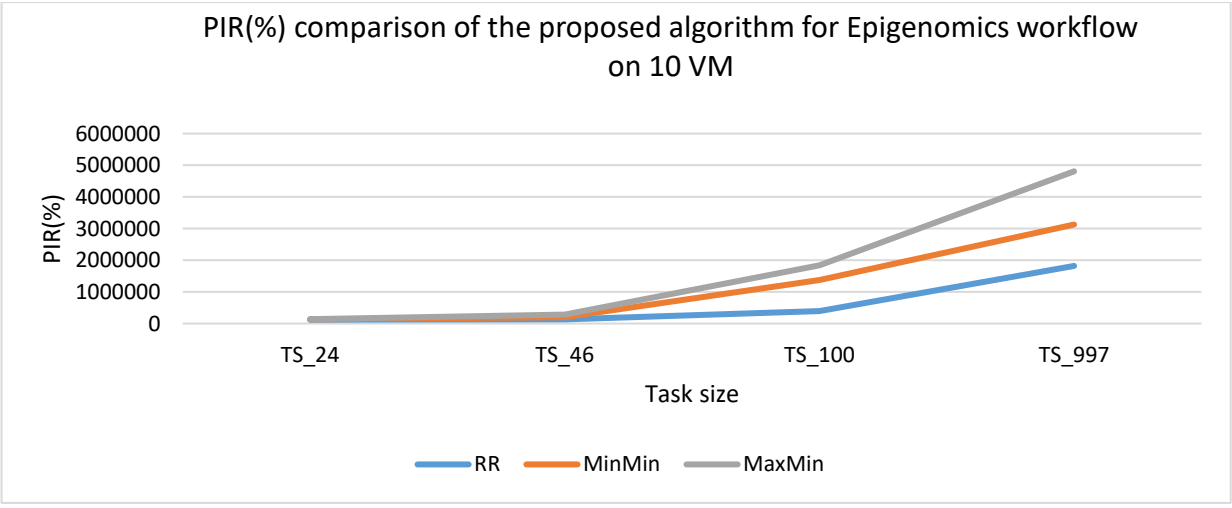


Figure 6. 16 PIR(%) comparison of the proposed algorithm for Epigenomics workflow on 10 VM

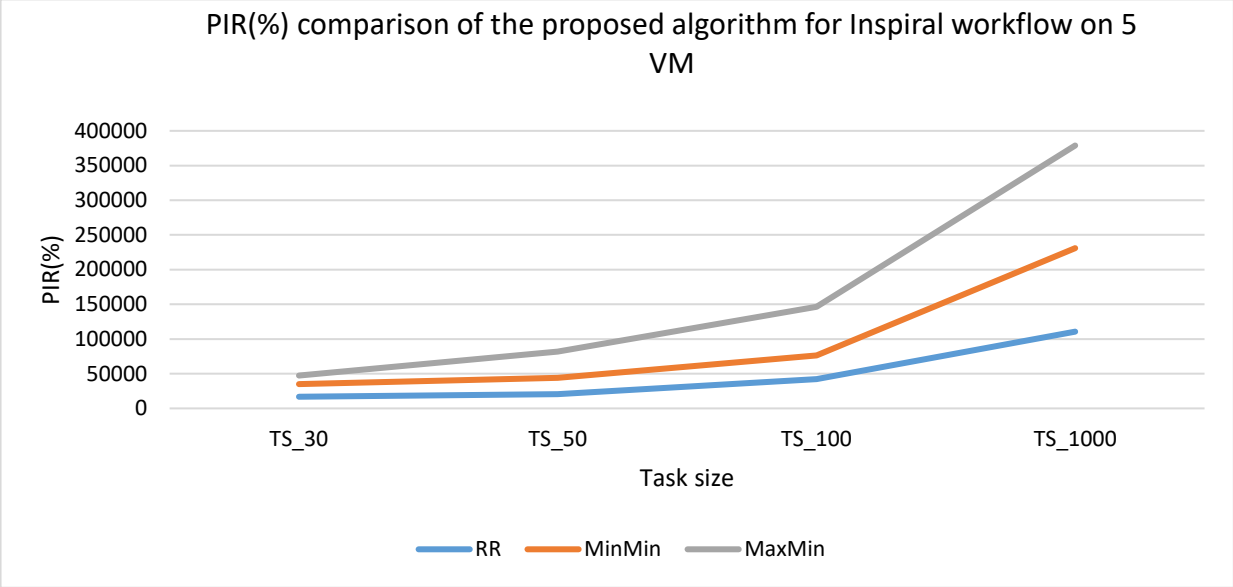


Figure 6. 17 PIR(%) comparison of the proposed algorithm for Inspiral workflow on 5 VM

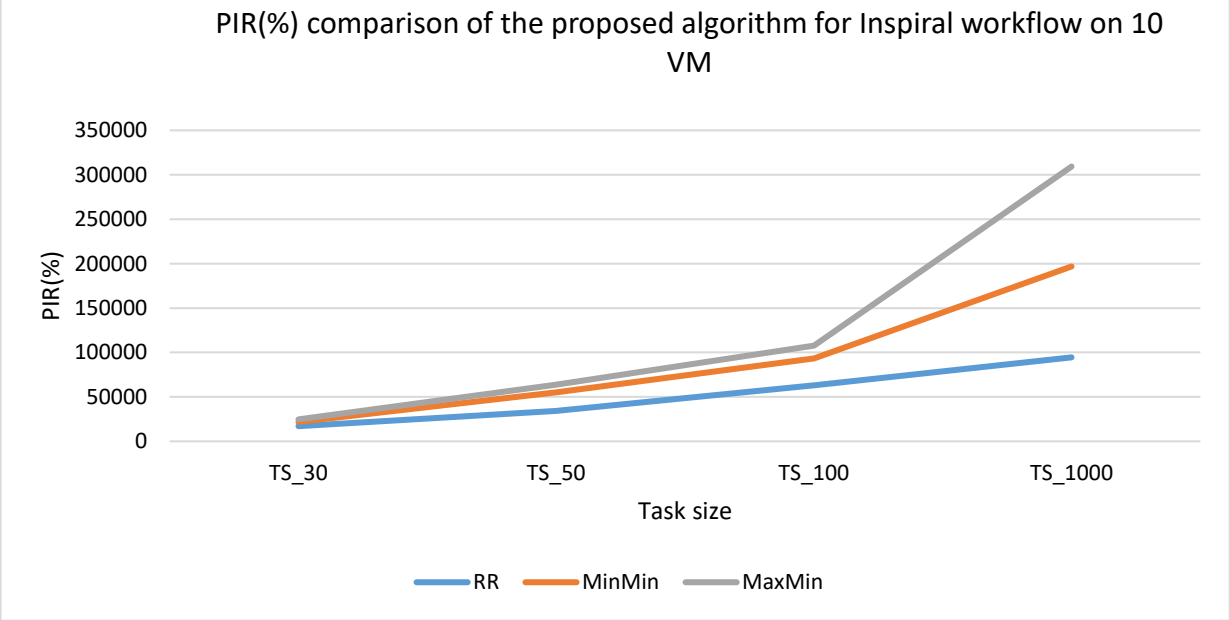


Figure 6. 18 PIR(%) comparison of the proposed algorithm for Inspiral workflow on 10 VM

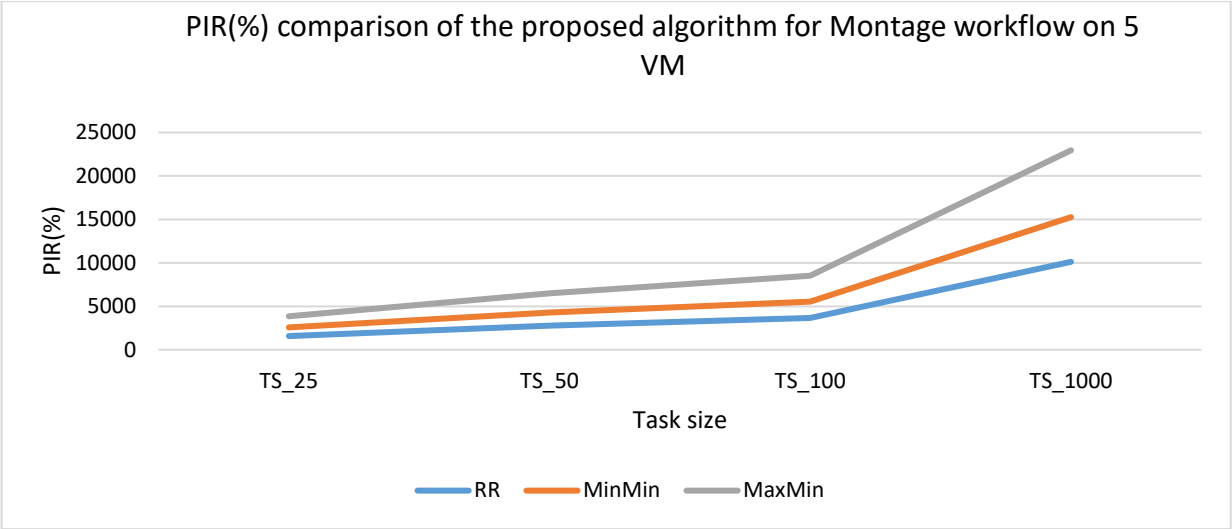


Figure 6. 19 PIR(%) comparison of the proposed algorithm for Montage workflow on 5 VM

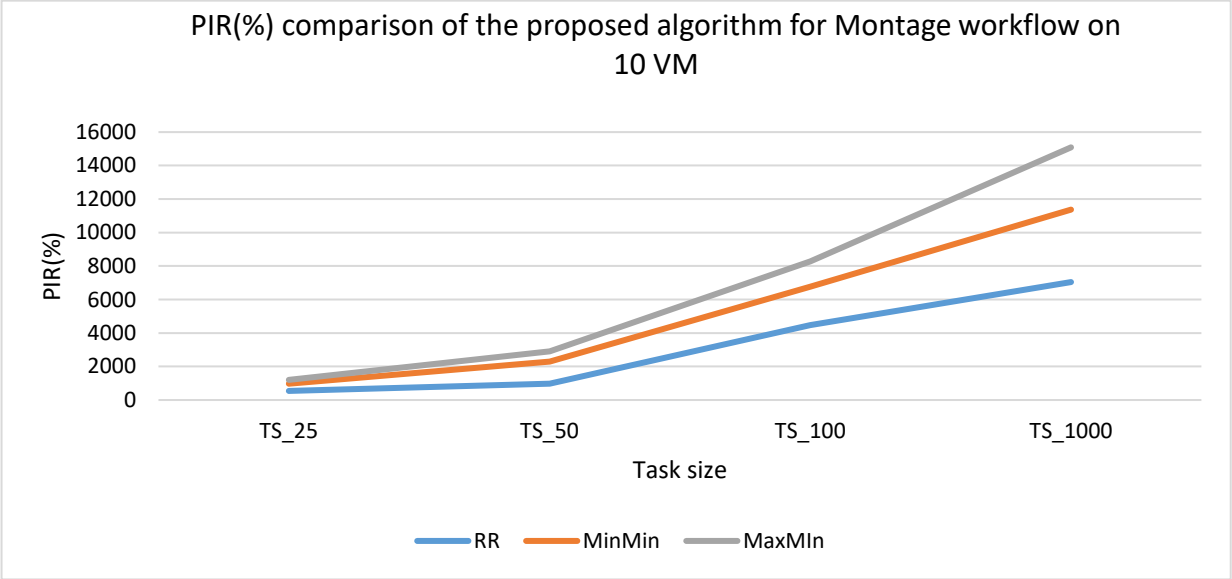


Figure 6. 20 PIR(%) comparison of the proposed algorithm for Montage workflow on 10 VM

CONCLUSION AND FUTURE WORK

Conclusion

Scientific workflows are gaining popularity as a natural means to process data. Workflows assist in the processing of rapidly rising volumes of data at significantly larger sizes, specifying data dependencies between the many components of an application. Scientific workflow management systems (SWfMS) provide tools for configuring, running, and monitoring them, allowing non-dependent processes to execute in parallel. Workflows were originally designed for high-performance computing systems, but they have also gained popularity on cloud platforms to take advantage of cloud-specific features such as elasticity and mobility. SWfMS runs on HPC-like cloud infrastructures, which are made up of virtual clusters with a shared file system located in the same data center. However, when the storage and processing requirements of large-scale workflows grow, and cloud providers impose quota limits, workflows must be distributed across multiple data centers (multisite workflows).

Scheduling of scientific workflows on cloud computing is an NP-complete problem that can be addressed from different perspectives. The primary goal of this research article was to experimentally determine the performance of the system if data movement is addressed in scheduling algorithms and the scheduling of data-intensive-scientific workflow scheduling on well-performing machines in cloud. Therefore, in this thesis work, we propose a single objective heuristic algorithm that is focused on optimizes the makespan for the data-intensive scientific workflows. This algorithm minimizes the makespan of the workflow based on the performance score of provided cloud machines. Score value of each machine is calculated based on certain performance metrics of each cloud resources like processing speed, number of processing cores, RAM size, bandwidth strength, and storage size. Tasks of the workflows are prioritized with the help of their computational cost, data transfer cost, upward and downward rank values which preserves tasks' topological order and their precedence constraints. Based on their final score value, machine with the highest value is mapped with high priority tasks on the TRQ.

Section 4.4 described the system architecture of the proposed method for scheduling scientific workflow on heterogeneous resources of cloud computing based on their performances. In contrast to previous research, our strategy suggests both the allocation and ordering of activities to be executed on cloud instances in order to reduce runtime and maximize locality by

guaranteeing that intermediate data products remain on cloud instances, which scheduled tasks that require them. By addressing these data movement and optimal task order problem of the scientific workflow, the entire makespan of the workflow scheduling is extremely minimized.

The proposed algorithm is compared to the default algorithm in WorkflowSim; RR, MinMin and MaxMin algorithm. The experimental study is conducted to compare the proposed algorithm performance in comparison with other heuristics in Section 5.2.2. Simulations and experiments on WorkflowSim are used to validate our implementation. The simulations use synthetic workflow data to describe medium to large-scale, computationally and data-intensive workflows with up to 1,000 jobs. In Section 6.1, the result that is found after the simulation is showed with the help of charts. Our findings reveal that, in all circumstances our VM performance based approach outperforms many heuristic scheduling algorithms like Round Robin, MinMin, and MaxMin. The proposed algorithm outperformed the RR, MinMin and MaxMin algorithm by 83.08%, 63.27% and 41.13% on 5 VMs respectively according to the result of CyberShake workflow with 30 tasks. It is demonstrated that the proposed approach performed better than RR, MinMin and MaxMin heuristics for Epigenomics workflow with 46 tasks. The proposed algorithm improved the makespan by 42.94%, 60.31% and 7.58% when compared to RR, MinMin and MaxMin respectively, on 5 VMs. RR performed worse than other heuristics for CyberShake and Montage workflows, in terms of the makespan. The proposed algorithm showed the result that lessen the makespan by 27.45% on computing nodes compared to RR. For MinMin and MaxMin, 33.24% and 1.76% respectively for Inspiral workflow with 30 task size.

In terms of makespan, the simulation results clearly show that the proposed method beats previous heuristic algorithms. With the exception of Montage, which has 1000 tasks for which MinMin outperforms the proposed approach by 0.15 %, the proposed algorithm's makespan improvements are quite evident for all other methods, as shown in fig 6.8. Multiple tasks in the same level with relatively short task execution times are assigned to various compute nodes with MinMin due to the parallel structure of these workflows. Given the fact that the lifespan of other methods fluctuates, the proposed algorithm functioned well from the start, with fewer VMs and as the number of processing nodes expanded. However, due of the multiple parallel structures of tasks in these workflows, the makespan of all algorithms, including the proposed approach, is elevated for Montage workflows for 30 computation nodes of 30.

This finding implies that the proposed scheduling technique would help cloud customers (especially scientific workflow users) in saving more money while using the cloud. This is due to the algorithm's ability to minimize the makespan time, which is the maximum completion time of tasks, allowing consumers to spend less time in the pay-per-use Cloud Computing environment.

FUTURE WORK

Despite the fact that scheduling is a discrete problem, there are only a few discrete multi-objective scheduling methods in the literature. A great deal of energy is consumed only to keep virtual machines running or to move data around without performing any important computation, in data-intensive application. As a result, reducing power consumption in the data center has a significant impact on the designing discrete versions of the newly proposed optimization algorithms will be critical in future investigations. Energy efficiency in cloud DCs has become extremely important due to the high cost of power usage and issues such as CO2 gas emissions. One of the fascinating strategies that can be advantageous in this scenario is dynamic power management. However, this proposed work did not consider the power consumption of the system. As a result, we suggest that future work should look at multi-objective scheduling with makespan improvement. In addition to this, another crucial issue to consider is reliability. Finding suitable method to identify the presence of malfunctioning VMs and problematic sites, on the other hand, is an essential issue that should be investigated in future studies.

***

This research was funded by Adama Science and Technology University with a grant number ASTU/SM-R/090/19.

Reference

.R, S., & .T, T. (2018). *How to Improve the Resource Utilization in Cloud Data Center?* 1–6. https://doi.org/10.1109/3ICT.2018.8855740

A, S., & K, G. (2016). A Review on Scheduling in Cloud Computing. *International Journal of UbiComp*, *7*, 9–15. https://doi.org/10.5121/iju.2016.7302

Aazam, M., & Huh, E. N. (2014). Framework of Resource Management for Intercloud Computing. *Mathematical Problems in Engineering*, *2014*. https://doi.org/10.1155/2014/108286

Abdulhamid, S., Shafie, A. L., Abdul-Salaam, G., & Madni, H. (2016). Secure Scientific Applications Scheduling Technique for Cloud Computing Environment Using Global League Championship Algorithm. *PLOS ONE*, *11*, e0158102. https://doi.org/10.1371/journal.pone.0158102

Abrishami, S., Naghibzadeh, M., & Epema, D. (2013). Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds. *Future Generation Computer Systems*, *29*, 158–169. https://doi.org/10.1016/j.future.2012.05.004

Arabnejad, H., & Barbosa, J. (2014). List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table. *IEEE Transactions on Parallel and Distributed Systems*, *25*, 682–694. https://doi.org/10.1109/TPDS.2013.57

Arabnejad, H., Barbosa, J., & Prodan, R. (2015). Low-time complexity budget-deadline constrained workflow scheduling on heterogeneous resources. *Future Generation Computer Systems*, *55*. https://doi.org/10.1016/j.future.2015.07.021

Ardagna, D., Casale, G., Ciavotta, M., Perez, J., & Wang, W. (2014). Quality-of-service in cloud computing: modeling techniques and their applications. *Journal of Internet Services and Applications*, *5*. https://doi.org/10.1186/s13174-014-0011-3

Arfeen, M., Pawlikowski, K., & Willig, A. (2011). A Framework for Resource Allocation Strategies in Cloud Computing Environment. *Proceedings - International Computer Software and Applications Conference*, 261–266. https://doi.org/10.1109/COMPSACW.2011.52

Aziza, H., & Krichen, S. (2020). A hybrid genetic algorithm for scientific workflow scheduling in cloud environment. *Neural Computing and Applications*, *32*, 1–16. https://doi.org/10.1007/s00521-020-04878-8

B, M. C., Marquez-chamorro, A. E., & Resinas, M. (2019). *A Cost-Effective Time-Constrained Multi-workflow Scheduling Strategy in Fog Computing* (Vol. 1, Issue 645751). Springer International Publishing. https://doi.org/10.1007/978-3-030-17642-6

Bahwaireth, K., Tawalbeh, L., Benkhelifa, E., Jararweh, Y., & Tawalbeh, M. (2016). Experimental Comparison of Simulation Tools for Efficient Cloud and Mobile Cloud Computing Applications. *EURASIP Journal on Information Security*, *2016*. https://doi.org/10.1186/s13635-016-0039-y

Batista, B., Estrella, J., Ferreira, C., Machado, D., Hideo, L., Reiff-Marganiec, S., Santana, M., & Helena, R. (2015). Performance Evaluation of Resource Management in Cloud Computing Environments. *3.234*, *10*. https://doi.org/10.1371/journal.pone.0141914

Bharathi, S., & Chervenak, A. (2009). *Data staging strategies and their impact on the execution of scientific workflows*. https://doi.org/10.1145/1552280.1592459

Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M.-H., & Vahi, K. (2008). Characterization of scientific workflows. *2008 3rd Workshop on Workflows in Support of Large-Scale Science, WORKS 2008*, 1–10. https://doi.org/10.1109/WORKS.2008.4723958

Bittencourt, Luiz F., Goldman, A., Madeira, E. R. M., Da Fonseca, N. L. S., & Sakellariou, R. (2018). Scheduling in distributed systems: A cloud computing perspective. *Computer Science Review*, *30*, 31–54. https://doi.org/10.1016/j.cosrev.2018.08.002

Bittencourt, Luiz Fernando, Sakellariou, R., & Madeira, E. (2010). DAG Scheduling Using a Lookahead Variant of the Heterogeneous Earliest Finish Time Algorithm. *Proceedings of the 18th Euromicro Conference on Parallel, Distributed and Network-Based Processing, PDP 2010*. https://doi.org/10.1109/PDP.2010.56

Brandic, I., & Dustdar, S. (2011). Grid vs Cloud — A Technology Comparison. *It - Information Technology*, *53*, 173–179. https://doi.org/10.1524/itit.2011.0640

Bryk, P., Malawski, M., Juve, G., & Deelman, E. (2016). Storage-aware Algorithms for Scheduling of Workflow Ensembles in Clouds. *Journal of Grid Computing*, *14*. https://doi.org/10.1007/s10723-015-9355-6

Bulaja, D., Bozic, K., Penevski, N., & Bacanin, N. (2019). *Introduction to Cloudsim*. 189–194. https://doi.org/10.15308/Sinteza-2019-189-194

Buyya, R., Ranjan, R., & Calheiros, R. (2009). Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challenges and Opportunities. *Computing Research Repository - CORR*, 1–11. https://doi.org/10.1109/HPCSIM.2009.5192685

Byrne, J., Svorobej, S., Giannoutakis, K. M., Tzovaras, D., Byrne, P. J., Östberg, P. O., Gourinovitch, A., & Lynn, T. (2017). A review of cloud computing simulation platforms & related environments. *CLOSER 2017 - Proceedings of the 7th International Conference on Cloud Computing and Services Science*, *Closer*, 651–663. https://doi.org/10.5220/0006373006790691

Casanova, H., Pandey, S., Oeth, J., Tanaka, R., da Silva, R., & Suter, F. (2018). *WRENCH: A Framework for Simulating Workflow Management Systems*. https://doi.org/10.1109/WORKS.2018.00013

Chana, I., & Singh, S. (2014). Quality of Service and Service Level Agreements for Cloud Environments: Issues and Challenges. In Z. Mahmood (Ed.), *Cloud Computing: Challenges, Limitations and R{\&}D Solutions* (Issue March 2018, pp. 51–72). Springer International Publishing. https://doi.org/10.1007/978-3-319-10530-7_3

Chen, W., & Deelman, E. (2012). WorkflowSim: A toolkit for simulating scientific workflows in distributed environments. *2012 IEEE 8th International Conference on E-Science, e-Science 2012*, 1–8. https://doi.org/10.1109/eScience.2012.6404430

Chen, W., Ferreira Da Silva, R., Deelman, E., & Sakellariou, R. (2015). Using imbalance metrics to optimize task clustering in scientific workflow executions. *Future Generation Computer Systems*, *46*, 69–84. https://doi.org/10.1016/j.future.2014.09.014

Chenni Kumaran, J., & Aramudhan, M. (2018). A survey on resource allocation strategies in

cloud. *International Journal of Reasoning-Based Intelligent Systems*, *10*(3–4), 328–336. https://doi.org/10.1504/IJRIS.2018.096229

Chopra, N., & Singh, S. (2013). Deadline and cost based workflow scheduling in hybrid cloud. *Proceedings of the 2013 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2013*, 840–846. https://doi.org/10.1109/ICACCI.2013.6637285

Cieza, E., Teylo, L., Frota, Y., Bentes, C., & Drummond, L. (2019). *A GPU-Based Metaheuristic for Workflow Scheduling on Clouds* (pp. 62–76). https://doi.org/10.1007/978-3-030-15996-2_5

Cottet, F., Delacroix, J., Kaiser, C., & Mammeri, Z. (2002). *Scheduling in Real-Time Systems*. https://doi.org/10.1002/0470856343

de Oliveira, D., Liu, J., & Pacitti, E. (2019). Data-Intensive Workflow Management: For Clouds and Data-Intensive and Scalable Computing Environments. *Synthesis Lectures on Data Management*, *14*, 1–179. https://doi.org/10.2200/S00915ED1V01Y201904DTM060

Derouiche, R., Brahmi, Z., & Gammoudi, M. (2019). *Cooperative Agents Based Data Placement Approach for Data Intensive Workflows*. 1–8. https://doi.org/10.1109/AICCSA47632.2019.9035227

Dong, F. (2009). *Workflow scheduling algorithms in the Grid*.

Duan, R., & Prodan, R. (2015). Cooperative Scheduling of Bag-of-Tasks Workflows on Hybrid Clouds. *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, *2015*, 439–446. https://doi.org/10.1109/CloudCom.2014.58

Duan, R., Prodan, R., & li, X. (2014). Multi-Objective Game Theoretic Schedulingof Bag-of-Tasks Workflows on Hybrid Clouds. *Cloud Computing, IEEE Transactions On*, *2*, 29–42. https://doi.org/10.1109/TCC.2014.2303077

Durillo, J., Nae, V., & Prodan, R. (2013). Multi-ObjectiveWorkflow Scheduling: An Analysis of the Energy Efficiency and Makespan Tradeoff. *Proceedings - 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2013*, 203–210. https://doi.org/10.1109/CCGrid.2013.62

Durillo, J., Prodan, R., & Barbosa, J. (2015). Pareto tradeoff scheduling of workflows on federated commercial Clouds. *Simulation Modelling Practice and Theory*, *58*. https://doi.org/10.1016/j.simpat.2015.07.001

Edward Jones. (n.d.). *Google Cloud Vs AWS in 2021*. Comparing the Giants. Retrieved June 13, 2021, from https://kinsta.com/blog/google-cloud-vs-aws/

Gill, S. S., & Chana, I. (2016a). A Survey on Resource Scheduling in Cloud Computing: Issues and Challenges. *Journal of Grid Computing*, *14*. https://doi.org/10.1007/s10723-015-9359-2

Gill, S. S., & Chana, I. (2016b). Cloud resource provisioning: survey, status and future research directions. *Knowledge and Information Systems*, *49*. https://doi.org/10.1007/s10115-016-0922-3

Gourisaria, M., Patra, S., & Khilar, P. (2018). *Energy Saving Task Consolidation Technique in Cloud Centers with Resource Utilization Threshold* (pp. 655–666).

https://doi.org/10.1007/978-981-10-6872-0_63

Groom, F. M. (2018). The Basics of Cloud Computing. In *Enterprise Cloud Computing for Non-Engineers*. https://doi.org/10.1201/9781351049221-1

Guruprasad, H. S., & H, bhavani. (2014). Resource Provisioning Techniques in Cloud Computing Environment: A Survey. *International Journal of Research in Computer and Communication Technology*, *3*, 395–401.

Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A., & Ullah Khan, S. (2015). The rise of "big data" on cloud computing: Review and open research issues. *Information Systems*, *47*, 98–115. https://doi.org/10.1016/j.is.2014.07.006

Hwang, K., Dongarra, J., & Fox, G. (2012). Virtual Machines and Virtualization of Clusters and Data Centers. *Computing*, 129–187.

Ismaeel, S., Miri, A., & Al-Khazraji, A. (2016). *Energy-Consumption Clustering in Cloud Data Centre*. https://doi.org/10.1109/ICBDSC.2016.7460373

Jagannatha, J. V. S. (2019). Task Scheduling and VM Allocation in Cloud Computing: A Survey. *International Journal of Distributed and Cloud Computing*, *7*(2), 7–18.

Jagirdar, S., Venkata, K., Reddy, S., & Qyser, D. (2013). CLOUD COMPUTING BASICS. *International Journal of Advanced Research in Computer and Communication Engineering*, *1*, 343.

Juve, G., Chervenak, A., Deelman, E., Bharathi, S., Mehta, G., & Vahi, K. (2013). Characterizing and profiling scientific workflows. *Future Generation Computer Systems*, *29*(3), 682–692. https://doi.org/10.1016/j.future.2012.08.015

Juve, G., & Deelman, E. (2010). Scientific workflows and clouds. *ACM Crossroads*, *16*, 14–18. https://doi.org/10.1145/1734160.1734166

Juve, G., & Deelman, E. (2008). *Resource Provisioning Options for Large-Scale Scientific Workflows*. 608–613. https://doi.org/10.1109/eScience.2008.160

Kale, V. (2017). Cloud Computing Basics. *Creating Smart Enterprises*, *June*, 141–171. https://doi.org/10.1201/9781315152455-6

Kaur, A., & Sengupta, J. (2017). Virtual Machine Scheduling using Improved Time Shared Policy in Cloud Computing. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, *6*(8), 1274–1277.

Kaur, G., & Mathai, K. J. (2018). Multi Criteria Rank Based Task Scheduling Algorithm for Scientific Workflows in IaaS Cloud Computing. *Proceedings of the International Conference on Inventive Research in Computing Applications, ICIRCA 2018*, 4849–4854. https://doi.org/10.1109/ICIRCA.2018.8597253

Keahey, K., & Freeman, T. (2008). Contextualization: Providing One-Click Virtual Clusters. *Proceedings - 4th IEEE International Conference on EScience, EScience 2008*, 301–308. https://doi.org/10.1109/eScience.2008.82

Kostaras, I., Drabo, C., Juneau, J., Reimers, S., Schröder, M., & Wielenga, G. (2020). *What Is Apache NetBeans* (pp. 3–28). https://doi.org/10.1007/978-1-4842-5370-0_1

Kumar, K., & Thaman, J. (2018). *Comparative Analysis of Workflow Scheduling Policies in Cloud Platforms* (pp. 267–275). https://doi.org/10.1007/978-981-10-5903-2_29

Kumar, M., Sharma, S. C., Goel, A., & Singh, S. P. (2019). A comprehensive survey for scheduling techniques in cloud computing. *Journal of Network and Computer Applications*, *143*(June), 1–33. https://doi.org/10.1016/j.jnca.2019.06.006

Lin, B., Zhu, F., Zhang, J., Chen, J., Chen, X., Xiong, N., & Lloret, J. (2019). A Time-Driven Data Placement Strategy for a Scientific Workflow Combining Edge Computing and Cloud Computing. *IEEE Transactions on Industrial Informatics*, *PP*, 1. https://doi.org/10.1109/TII.2019.2905659

Lin, W., Wu, W., & Wang, J. (2016). A Heuristic Task Scheduling Algorithm for Heterogeneous Virtual Clusters. *Scientific Programming*, *2016*, 1–10. https://doi.org/10.1155/2016/7040276

Liu, J., Pacitti, E., Valduriez, P., & Mattoso, M. (2015). A Survey of Data-Intensive Scientific Workflow Management. *Journal of Grid Computing*, *13*. https://doi.org/10.1007/s10723-015-9329-8

Liu, Z., Qu, W., Liu, W., Li, Z., & Xu, Y. (2014). Resource preprocessing and optimal task scheduling in cloud computing environments. *Concurrency and Computation: Practice and Experience*, *27*. https://doi.org/10.1002/cpe.3204

Lizhen, L., Zhang, J., Yue, L., Shi, Y., Li, H., & Yuan, D. (2015). A Genetic Algorithm Based Data Replica Placement Strategy for Scientific Applications in Clouds. *IEEE Transactions on Services Computing*, *PP*, 1. https://doi.org/10.1109/TSC.2015.2481421

Mann, Z. (2015). Allocation of Virtual Machines in Cloud Data Centers—A Survey of Problem Models and Optimization Algorithms. *ACM Computing Surveys*, *48*, 1–34. https://doi.org/10.1145/2797211

Mansouri, N., Ghafari, R., & Zade, B. M. H. (2020). Cloud computing simulators: A comprehensive review. *Simulation Modelling Practice and Theory*, *104*(February). https://doi.org/10.1016/j.simpat.2020.102144

Marinescu, D. C. (2018). Cloud Resource Management and Scheduling. In *Cloud Computing*. https://doi.org/10.1016/b978-0-12-812810-7.00012-1

Marozzo, F., Rodrigo Duro, F., Garcia Blas, J., Carretero, J., Talia, D., & Trunfio, P. (2017). A data-aware scheduling strategy for workflow execution in clouds. *Concurrency Computation* , *29*(24). https://doi.org/10.1002/cpe.4229

Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J., & Ghalsasi, A. (2011). Cloud computing - The business perspective. *Decision Support Systems*, *51*(1), 176–189. https://doi.org/10.1016/j.dss.2010.12.006

Mastroeni, L., Mazzoccoli, A., & Naldi, M. (2019). Service Level Agreement Violations in Cloud Storage: Insurance and Compensation Sustainability. *Future Internet*, *11*, 142. https://doi.org/10.3390/fi11070142

Mathew, M. (2018). *Virtualization and Scheduling In Cloud Computing Environment@_ A Study*.

Mimura Gonzalez, N., Carvalho, T., & Miers, C. (2017). Cloud resource management: towards efficient execution of large-scale scientific applications and workflows on complex infrastructures. *Journal of Cloud Computing*, *6*. https://doi.org/10.1186/s13677-017-0081-4

Mohamed Ridha, B., Soltani, A., Bouhank, A., & Daoudi, M. (2018). *New Search Based Methods to Solve Workflow Scheduling Problem in Cloud Computing*. 647–652. https://doi.org/10.1109/CoDIT.2018.8394855

Nasonov, D., Visheratin, A., Butakov, N., Shindyapina, N., Melnik, M., & Boukhanovsky, A. (2017). Hybrid evolutionary workflow scheduling algorithm for dynamic heterogeneous distributed computational environment. *Journal of Applied Logic*, *24*, 50–61. https://doi.org/10.1016/j.jal.2016.11.013

Negru, C., Mocanu, M., Cristea, V., Sotiriadis, S., & Bessis, N. (2017). Analysis of power consumption in heterogeneous virtual machine environments. *Soft Computing*, *21*.

O'Loughlin, J., & Gillam, L. (2014). Good Performance Metrics for Cloud Service Brokers. *CLOUD 2014*.

Peter Mell (NIST); Tim Grance (NIST). (n.d.). The NIST Definition of Cloud Computing. *NIST Special Publication 800-145*, *728*, 269–274. http://www.nist.gov/manuscript-publication-search.cfm?pub_id=909494.%0A

Rajkumar Buyya, Christian Vecchiola, S. T. S. (20143). *Mastering Cloud Computing Chapter 1 - Introduction* (S. T. S. Rajkumar Buyya, Christian Vecchiola (ed.)). Morgan Kaufmann,. https://doi.org/10.1016/B978-0-12-411454-8.00001-2.

Rana, A., & Tanwar, G. (2014). *Exemplifying Practical issues Of Resource Management in Cloud Computing*. *3*(3), 680–683.

Rodriguez, M. A., & Buyya, R. (2017). A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments. *Concurrency Computation* , *29*(8), 1–23. https://doi.org/10.1002/cpe.4041

S., S. (2011). *Efficient Task Scheduling Algorithms for Cloud Computing Environment*. https://doi.org/10.1007/978-3-642-22577-2_11

Shea, R., Wang, H., & Liu, J. (2014). A deep investigation into network performance in virtual machine based cloud environments. *Proceedings - IEEE INFOCOM*, 1285–1293. https://doi.org/10.1109/INFOCOM.2014.6848061

Shibata, T., Choi, S. J., & Taura, K. (2010). File-access patterns of data-intensive workflow applications and their implications to distributed filesystems. *HPDC 2010 - Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, 746–755. https://doi.org/10.1145/1851476.1851585

Singh, R., & Singh, S. (2013). Score Based Deadline Constrained Workflow Scheduling Algorithm for Cloud Systems. *International Journal on Cloud Computing: Services and Architecture*, *3*(6), 31–41. https://doi.org/10.5121/ijccsa.2013.3603

Singh, V., Gupta, I., & Jana, P. K. (2019). An Energy Efficient Algorithm for Workflow Scheduling in IaaS Cloud. *Journal of Grid Computing*. https://doi.org/10.1007/s10723-019-09490-2

Sukhoroslov, O. (2019). *An Experimental Study of Data Transfer Strategies for Execution of Scientific Workflows* (pp. 67–79). https://doi.org/10.1007/978-3-030-25636-4_6

Teylo, L., de Paula, U., Frota, Y., de Oliveira, D., & Drummond, L. M. M. A. (2017). A hybrid evolutionary algorithm for task scheduling and data assignment of data-intensive scientific

workflows on clouds. *Future Generation Computer Systems*, *76*, 1–17. https://doi.org/10.1016/j.future.2017.05.017

Toan, P., Nguyen The, L., Cuong, D., & Long, Đ. (2015). A particle swarm optimization-based workflow scheduling algorithm in the cloud computing enveronment. *Journal of Science, Natural Science*, *60*, 47–55. https://doi.org/10.18173/2354-1059.2015-0007

Topcuoglu, H., Hariri, S., & Wu, M.-Y. (2002). Performance-effective and low-complexity task scheduling forheterogeneous computing. *Parallel and Distributed Systems, IEEE Transactions On*, *13*, 260–274. https://doi.org/10.1109/71.993206

Ul Islam, M. S., & Rana, B. (2017). *Task Scheduling in Cloud Computing*.

Vaezi, M., & Zhang, Y. (2017). *Cloud Mobile Networks*. https://doi.org/10.1007/978-3-319-54496-0

Varghese, B., Akgün, Ö., Miguel, I., Thai, L., & Barker, A. (2016). Cloud Benchmarking For Maximising Performance of Scientific Applications. *IEEE Transactions on Cloud Computing*, *7*. https://doi.org/10.1109/TCC.2016.2603476

Vijaya, C., & Srinivasan, P. (2016). A survey on resource scheduling in cloud computing. *International Journal of Pharmacy and Technology*, *8*(4), 26142–26162.

Wang, M., Zhang, J., Dong, F., & Luo, J. (2014). *Data Placement and Task Scheduling Optimization for Data Intensive Scientific Workflow in Multiple Data Centers Environment*. 77–84. https://doi.org/10.1109/CBD.2014.19

Yang, Y. (2013). Heuristic Scheduling Algorithms for Allocation of Virtualized Network and Computing Resources. *Journal of Software Engineering and Applications*, *06*, 1–13. https://doi.org/10.4236/jsea.2013.61001

Zhan, Z.-H., Liu, X.-F., Gong, Y.-J., Zhang, J., Chung, H., & Li, Y. (2015). Cloud Computing Resource Scheduling and a Survey of Its Evolutionary Approaches. *ACM Computing Surveys*, *47*, 1–33. https://doi.org/10.1145/2788397

Zhang, Q., Zhani, M. F., Boutaba, R., & Hellerstein, J. (2014). Dynamic Heterogeneity-Aware Resource Provisioning in the Cloud. *IEEE Transactions on Cloud Computing*, *2*. https://doi.org/10.1109/TCC.2014.2306427