Budget-constrained Workflow Applications Scheduling in Workflow-as-a-Service Cloud Computing Environments

Muhammad Hafizhuddin Hilman

Submitted in total fulfilment of the requirements of the degree of

Doctor of Philosophy

School of Computing and Information Systems
THE UNIVERSITY OF MELBOURNE

March 2020

ORCID: 0000-0003-2772-9216

Budget-constrained Workflow Applications Scheduling in Workflow-as-a-Service Cloud Computing Environments

Muhammad Hafizhuddin Hilman

Principal Supervisor: Prof. Rajkumar Buyya

Co-Supervisor: Dr. Maria Rodriguez Read

Abstract

The adoption of workflow, an inter-connected tasks and data processing application model, in the scientific community has led to the acceleration of scientific discovery. The workflow facilitates the execution of complex scientific applications that involves a vast amount of data. These workflows are large-scale applications and require massive computational infrastructures. Therefore, deploying them in distributed systems, such as cloud computing environments, is a necessity to acquire a reasonable amount of processing time.

With the increasing demand for scientific workflows execution and the rising trends of cloud computing environments, there is a potential market to provide a computational service for executing scientific workflows in the clouds. Hence, the term Workflow-as-a-Service (WaaS) emerges along with the rising of the Everything-as-a-Service concept. This WaaS concept escalates the functionality of a conventional workflow management system (WMS) to serve a more significant number of users in a utility service model. In this case, the platform, which is called the WaaS platform, must be able to handle multiple workflows scheduling and resource provisioning in cloud computing environments in contrast to its single workflow management of traditional WMS.

This thesis investigates the novel approaches for budget-constrained multiple workflows resource provisioning and scheduling in the context of the WaaS platform. They address the challenges in managing multiple workflows execution that not only comes from the users' perspective, which includes the heterogeneity of workloads, quality of services, and software requirements, but also problems that arise from the cloud environments as the underlying computational infrastructure. The latter aspect brings up the issues of the heterogeneity of resources, performance variability, and uncertainties in the form of overhead delays of resource provisioning and network-related activities. It pushes a boundary in the area by making the following contributions:

1. A taxonomy and survey of the state-of-the-art multiple workflows scheduling in multi-tenant distributed computing systems.

2. A budget distribution strategy to assign tasks' budgets based on the heterogeneous type of VMs in cloud computing environments.

3. A budget-constrained resource provisioning and scheduling algorithm for multiple workflows that aims to minimize workflows' makespan while meeting the budget.

4. An online and incremental learning approach to predict task runtime that considers the performance variability of cloud computing environments.

5. The implementation of multiple workflows scheduling algorithm and its integration to extend the existing WMS for the development of WaaS platform.

Declaration

This is to certify that

1. the thesis comprises only my original work towards the PhD,
2. due acknowledgement has been made in the text to all other materials used,
3. the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies, and appendices.

Muhammad Hafizhuddin Hilman, March 2020

This page intentionally left blank.

Preface

This thesis research has been carried out in the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, Melbourne School of Engineering, The University of Melbourne. The main contributions of the thesis are discussed in Chapters 2–6 and are based on the following publications:

- **Muhammad H. Hilman**, Maria A. Rodriguez, and Rajkumar Buyya. ‘Multiple Workflows Scheduling in Multi-tenant Distributed Systems: A Taxonomy and Future Directions.‘ *ACM Computing Surveys (CSUR), vol. 53, no. 1*, Pages 10:1-10:39, 2020.

- **Muhammad H. Hilman**, Maria A. Rodriguez, and Rajkumar Buyya. ‘Task-based Budget Distribution Strategies for Scientific Workflows with Coarse-grained Billing Periods in IaaS Clouds.‘ *In Proceedings of the 13th IEEE International Conference on e-Science (e-Science)*, Pages 128-137, 2017.

- **Muhammad H. Hilman**, Maria A. Rodriguez, and Rajkumar Buyya. ‘Resource-sharing Policy in Multi-tenant Scientific Workflow-as-a-Service Platform.‘ *Journal of Computer and System Sciences (JCSS)*, 2020 (under review).

- **Muhammad H. Hilman**, Maria A. Rodriguez, and Rajkumar Buyya. ‘Task Runtime Prediction in Scientific Workflows Using an Online Incremental Learning Approach.‘ *In Proceedings of the 11th ACM/IEEE International Conference on Utility and Cloud Computing (UCC)*, Pages 93-102, 2018.

- **Muhammad H. Hilman**, Maria A. Rodriguez, and Rajkumar Buyya. ‘Workflow-as-a-Service Cloud Platform and Deployment of Bioinformatics Workflow Applications.‘ *Knowl-*

edge Management in Development of Data-Intensive Software Systems (KMDDIS), 2020 (under review).

Acknowledgements

I would like to thank my supervisor, Professor Rajkumar Buyya, for allowing me to pursue this PhD in CLOUDS Laboratory. I am very grateful for his guidance, advice, and motivation throughout my candidature. I am also profoundly grateful for my co-supervisor, Dr. Maria Rodriguez Read for her support in my journey, especially her guidance in solving research technicalities.

I would like to express my gratitude to the PhD committee members, Professor Wally Smith, for his constructive comments and guidance during my candidature. I would also like to thank all the past and current members of the CLOUDS Laboratory, at the University of Melbourne for their friendship and support. In particular, I thank Dr. Rodrigo Calheiros, Dr. Amir Vahid, Dr. Adel Nadjaran Toosi, Dr. Sukhpal Gill, Dr. Chenhao Qu, Dr. Yasser Mansouri, Dr. Jungmin Son, Dr. Bowen Zhou, Dr Xunyun Liu, Dr. Minxian Xu, Dr. Safiollah Heidari, Caesar Wu, Sara Kardani Moghaddam, Redowan Mahmud, Tawfiqul Islam, Shashikant Ilager, TianZhang He, Mohammad Goudarzi, Mohammadreza Razian, Zhiheng Zhong, Samodha Palewatta, and Amanda Jayanetti.

I acknowledge the Republic of Indonesia and the Indonesia Endowment Fund for Education (LPDP) for providing me with scholarships to pursue my doctoral studies.

I would like to thank my parents and my sisters for their loving support. The distance does not hinder their support for my voyage in Australia. I also thank my parents- and family-in-law for being supportive away from home. Finally, I thank my wife for her selflessness and dedication to support my PhD. For my kids, Shafiyah, Hamka, and Hanina, I am proud to have your laughter and happiness accompanying the journey.

Muhammad Hafizhuddin Hilman
Melbourne, Australia
March 2020

To my beloved wife, Susan Adella, and the lovely {Shafiyah, Hamka, Hanina} Hafizh.

Without them, this journey would not be as lively as it is.

This page intentionally left blank.

Contents

This page intentionally left blank.

List of Figures

List of Tables

Chapter 1

Introduction

The concept of workflow has evolved from the manufacturing and business process fields to a broader notion representing a structured process flow design. Within the scientific community, workflows refer to a model for automating and managing the distributed execution of a complex problem that requires the flow of data between different applications [1]. Many scientific problems adopt this model to overcome the limitations of a single scientific application to process a vast amount of data. Examples include Scientific Workflows [2] that consist of HPC applications for e-Science and MapReduce Workflows [3] that are used to process Big Data analytics. Such workflows are large-scale distributed applications and require extensive computational resources to execute in a reasonable amount of processing time. Therefore, workflows are commonly deployed on distributed systems, in particular, cluster, grid, and cloud computing environments.

The advent of multi-tenant environments like clouds and the shifting trend from the traditional on-premises to the utility computing era has led to the emergence of platforms that offer workflows processing as a service. These platforms continuously receive many requests for workflow executions from different users along with their various Quality of Service (QoS) requirements. The provider must then be able to schedule these workflows in a way that each of their requirements is fulfilled. A simple way to achieve this is by allocating a set of dedicated resources to execute each workflow. However, the inter-dependent tasks produce unavoidable idle gaps in the schedule. Hence, dedicating a set of resources for each user can be considered inefficient in environments where multiple workflows are involved, since it leads to resources being underutilized. This approach, in turn, may cause a significant loss for the providers that generate revenue from the utilization of resources. Consequently, the strategies implemented in such platforms should aim to improve resource utilization while still complying with the unique requirements of different users.

Accommodating multi-tenants with different requirements creates a highly complex system. The first problem lies in how the system handles various workflow applications. A variety of applications involve different software libraries, dependencies, and hardware requirements. The users should be able to customize the configurations along with their QoS when submitting the workflows. Furthermore, multi-tenant systems must have a general approach to handle different types of computational requirements from different workflows. Another consideration related to multi-tenancy is the strategy to maintain fairness between multiple users that should be achieved through prioritization in scheduling and automatic scaling of the resources. The last aspect that should be noticed in multi-tenant platforms is the performance variability of computational resources. It is known that virtualization-based infrastructures might encounter performance degradation due to multi-tenancy, virtualization overhead, geographical location, and temporal aspects [4].

This thesis addresses problems in managing a platform for executing workflow as a service in cloud environments referred to as Workflow-as-a-Service (WaaS). It focuses on designing novel resource provisioning and scheduling strategies to handle multiple users that submit their workflows' jobs continuously in a near real-time scenario. Therefore, the algorithms must take advantage of the cloud resources that can be elastically provisioned on-demand using a pay-as-you-go model. At the same time, the clouds encounter inherent uncertainties in performance, which must also be considered by the resource management algorithms.

The research objectives are achieved by conducting a study resulting in a detailed taxonomy and comprehensive survey based on the state-of-the-art multiple workflow scheduling algorithms. In addition to the study, a dynamic resource provisioning and scheduling algorithm–includes its budget distribution strategy–that enforces a resource-sharing policy to increase the resource efficiency when handling multiple workflows, is proposed. It also introduces an online incremental learning approach for scientific workflows' resource consumption estimation and runtime prediction, as a pre-requisite to the scheduling. Finally, an existing workflow management system was extended to support the WaaS platform development. We present its capabilities with a case study using various real-life scientific applications from the bioinformatics fields.

1.1 Background

This section presents an overview of the concepts which drive the research problem addressed in the thesis. They are multi-tenancy in cloud computing and scientific workflow as a service.

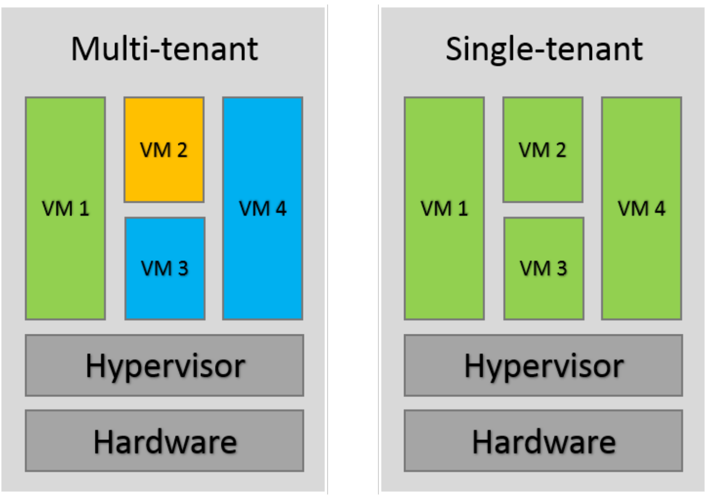


Fig. 1.1: Multi-tenant vs single-tenant clouds

1.1.1 Multi-tenancy in Cloud Computing

Cloud computing simplifies management of computational resources with its virtualization technology. In this case, virtualization creates an abstraction layer that hides the underlying software and hardware complexity of a cloud data center, which enables resource allocation, consolidation, and auto-scaling [5]. From the perspective of cloud providers, this virtualization technology expedites further commercialization of cloud resources. The providers can slice their infrastructure into a smaller allocation of resources easily and lease it to the users with a variety of computational configurations (i.e., VM types). On the other hand, from the users' point of view, virtualization establishes an illusion that they appear to run on a dedicated computer system. Therefore, it is not uncommon for a particular cloud computing user to utilize the same underlying cloud hardware infrastructures with hundreds or thousands of other users without any performance dubiety.

Based on this virtualization model, cloud computing is naturally a multi-tenant environment except, of course, the private clouds, which may lead to a different assumption. Cloud computing providers market their infrastructure and price the computational unit based on many intrinsic and extrinsic factors [6]. The intrinsic involves the internal hardware infrastructure, such as the utilization rate and the energy efficiency, against the extrinsic, which represents the users and their willingness to pay. The behaviour of these two opposite factors profoundly influences the decision of cloud computing providers to price their products. The illustration of virtualization and multi-tenancy in clouds is depicted in Fig. 1.1.

In this multi-tenant environment, cloud computing providers expect to gain a high data center utilization rate to create revenue. On the other hand, the users hope for the promised computational performance based on the service level agreements (SLA) regardless of the sharing of computational infrastructure. However, multi-tenancy in cloud computing environments creates a complicated system that results in an inevitable performance variability, as reported by Leitner and Cito [4]. For example, it is not uncommon for the clouds to exhibit performance degradation at a particular time when the number of requests is at its highest point (i.e., peak-hours).

The multi-tenancy in cloud computing environments also drives the cloud providers to diversify their products' heterogeneity to serve various types of users. The cloud providers offer a bundle of computational capacity in the form of a virtual machine type to serve the different needs of users. This variety includes CPU speed, memory size, bandwidth capacity, GPU-enabled, and storage type. However, these variants are within the border of virtualization. Hence, these virtualization-based products are impacted by inevitable overhead and performance degradation that are inherently embedded within the technology.

To serve the users that cannot afford the substantial cost of virtualization overhead, the cloud providers offer another type of resource. Examples of products include the bare-metal server, which practically eliminates virtualization, or lightweight virtualization products that maintain isolation in the form of microservices such as containers. Recently, the cloud providers also launched function as a service, providing even higher-level of abstraction for code execution on-demand. In this scenario, users do not even need to think about infrastructure management.

The heterogeneity of resources is an inevitable consequence of the multi-tenancy aspect of cloud computing environments. In serving different users, the cloud providers encounter different needs and requirements that become an essential factor in the creation of the product. In this thesis, we are interested in exploring the opportunities and challenges in cloud computing environments that have a significant impact on how the resource management policies must be designed to solve the problems inherently caused by its multi-tenant architecture and characteristics.

1.1.2 Scientific Workflow as a Service

Scientific workflows are widely used to automate scientific experiments. The latest detection of gravitational waves by the LIGO project [7] is an example of a scientific breakthrough assisted by workflow technologies. These workflows are composed of multiple tasks and dependencies that represent the flow of data between them. Scientific workflows are usually large-scale applications that require extensive computational resources to process. As a result, distributed systems with an abundance of storage, network, and computing capacity are widely used to deploy these resource-intensive applications.

The complexity of scientific workflows execution urges scientists to rely on workflow management systems (WMS), which manage the deployment of workflows in distributed resources. Their main functionalities include but are not limited to scheduling the workflows, provisioning the resources, managing the dependencies of the tasks, and staging the input and output data. A fundamental responsibility of WMS and the focus of this work is the scheduling of workflow tasks. In general, this process consists of two stages, i) mapping the execution of tasks on distributed resources and ii) acquiring and allocating the appropriate compute resources to support them. Both of these processes need to be carried out, while considering the QoS requirements specified by users and preserving the dependencies between tasks. These requirements make the workflow scheduling process a challenging problem.

WMS technology has consistently evolved along with the emergence of multi-tenant distributed systems. ASKALON [8], CloudBus [9], HyperFlow [10], Kepler [11], Pegasus [12], and Taverna [13] are examples of WMS that have been continuously developed within the scientific community. In the era of cluster and grid environments, the main features of the WMS focused on the workflow's representation, utilizing web technology, and providing a Graphical User Interface for the users. Then, the requirements shifted to the fulfilment of user-defined QoS in market-oriented cloud computing environments, followed by the need to make workflow's execution more lightweight using various microservices technology.

The advancement of e-Science infrastructure in the form of scientific applications empowers a large number of scientists around the world to start the shifting trend of scientific experiments. They are part of a broader community that is called the *Long Tail of Science*. The smaller group of scientists that run their scientific experiments on a far smaller scale than the LIGO project but generating a more significant number of scientific data and findings [14]. However, not many scientists can afford to build their dedicated computational infrastructure for their experiments. In this case, there is a potential market for providing such services to the scientific community.

WaaS is an emerging paradigm that offers the execution of scientific workflow as a service. The service provider lies either in the Platform-as-a-Service (PaaS) or Software-as-a-Service (SaaS) layer based on the cloud service model. WaaS providers make use of distributed computational resources to serve the enormous need for computing power in the execution of scientific workflows. They provide a holistic service to scientists starting from the user interface in the submission portal, applications installation, and configuration, staging of input/output data, workflow scheduling, and resource provisioning. WaaS platforms are designed to process multiple workflows from different users. The workload is expected to arrive continuously, and workflows must be handled as soon as they arrive due to the QoS constraints defined by the users. Therefore, WaaS platforms must deal with a high level of complexity derived from their multi-tenant and dynamic nature, contrary to a traditional WMS that is commonly used for managing a single workflow execution.

Several variations of WaaS framework–which extend the WMS–are found in the literature. Wang et al. [15] described a service-oriented architecture that separates the components into user management, scheduler, storage, and VM management layer. Meanwhile, a framework with a similar division that emphasized on distributed storage was proposed by Esteves and Veiga [16]. Another architecture for multi-tenant scientific workflows execution in clouds emplaced the proposed framework as a service layer above the Infrastructure-as-a-Service (IaaS) [17]. In general, we identified three primary layers in WaaS platforms: the tenant layer, the scheduler layer, and the resource layer. Based on the identified requirements of WaaS platforms, we propose an architecture for this system focusing on the scheduler component, as depicted in Fig. 1.2.

Firstly, the tenant layer manages the workflows submission, where users can configure their preferences and define the QoS of their workflows. The scheduler layer is responsible for placing the tasks on either existing or newly acquired resources and consists of four components: task runtime estimator, task scheduler, resource provisioner, and resource monitor. The task runtime estimator is used to predict the amount of time a task will take to complete within a specific computational resource. Another component, the task scheduler, is used to map a task into a selected virtual machine for execution. Meanwhile, resource provisioner is used to acquire and release virtual machines from third-party providers. The resource monitor is used to collect the resource consumption data of a task executed in a particular virtual machine. These data are stored

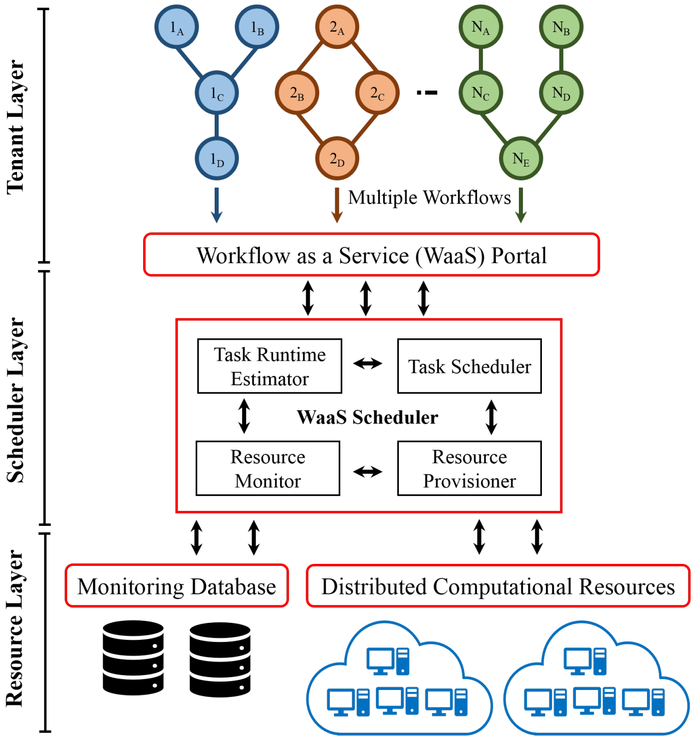


Fig. 1.2: Workflow-as-a-service architecture

in a monitoring database and are used by the task runtime estimator to build a model to estimate the task's runtime. The third-party infrastructure, with which the platforms interact, fall into the resource layer. In this thesis, the investigation is based on the assumption of the WaaS platform and its relations to multi-tenant aspects in cloud computing environments.

1.2 Motivation

The latest report by Gartner [18] presented public cloud computing trends in 2020 and its forecast for the next five years. It is expected that 60% of organizations all over the world will migrate their computational load to external cloud providers by 2022 and this number is expected to rise as high as 90% by 2025. This means that within five years, almost all services will be hosted on clouds. In addition, Gartner estimated that the revenue of cloud providers would grow to US$266.4 billion (A$387.83 billion) in 2020.

Furthermore, the popularity of using workflows to automate and manage complex scientific applications has risen as this increasing trend of migrating computational load from the on-premises datacenter to the cloud computing environments. This demand drives the market to provide many kinds of computational infrastructure as a service, including but not limited to compute resources, platforms, and the software. Therefore, there is a potential market to provide the execution of workflows as the utility services for the scientific community.

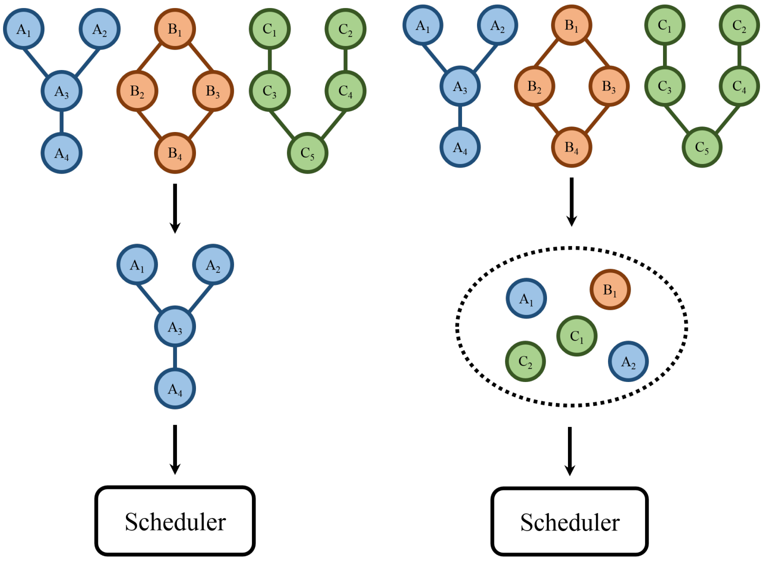
The existing WMSs are designed to manage the execution of a single workflow. As platform providing services to multiple users, they must incorporate specific design principles to take advantage of the opportunities and handle the challenges derived from the multi-tenancy aspect. In this case, we argue that an approach must be designed to manage the workflow's execution different to the traditional way in which current WMS are capable of, as discussed in Section 1.1.2. To the best of our knowledge, this kind of platform is still in progress, and no working system has provided this particular service to the market yet.

Various theoretical works address the problems in designing the WaaS platform [15] [16]. Nevertheless, there still is a large gap to fill, and room for improvement for as many issues have not been considered in the existing solutions. Two primary keywords in this research are multi-tenancy and heterogeneity. There is an abundance of problem derivations that need to be solved related to those keywords which apply to two different actors, i) the WaaS cloud providers and its relations with the cloud computing environments as their backbone services, and ii) the users and their various workflow applications, which imply a diverse spectrum of the requirements.

The progress of the WaaS platform development and the research gap in this area motivates us to undertake this research. Based on our expertise and experience, we choose to contribute to one particular aspect of resource provisioning and scheduling area, while also considering other elements related to the development of the WaaS platform.

1.3 Problem Definition: Scheduling Multiple Workflows

The scope of this thesis is limited to the multiple workflows that are modelled as directed acyclic graphs (DAGs) where a workflow W consists of a set of tasks $T = (t_1, t_2, \ldots, t_n)$ and a set of directed edges $E = (e_{12}, e_{13}, ..., e_{mn})$ in which an edge e_{ij} represents a data dependency between task t_i (parent task) and task t_j (child task). Hence, t_j will only be ready for execution after t_i has completed. The workflow is completed the execution when the exit task t_{exit} (i.e., task without successors) is finished being processed. The purpose of DAG scheduling is allocating



(a) Independent scheduling (b) Simultaneous scheduling

Fig. 1.3: Two approaches in scheduling multiple workflows

the tasks to computational resources in a way that the precedence constraints among the tasks are preserved. Specifically, the workflows submitted to multi-tenant computing platforms belong to different users and are not necessarily related to each other. As a result, heterogeneity becomes a defining characteristic of the workload that covers various aspects of workflows, including the type of applications, the size of the workflows, and the user-defined QoS.

When these workflows arrive into the platforms, planning the schedule by exploiting the information (i.e., topological structure, computational requirement, size, input) as being implemented in a static workflow scheduling is not always beneficial. For example, a strategy of partitioning tasks before runtime to minimize the data transfer is proven to be efficient for data-intensive workflows [19]. This strategy can be adopted in multi-tenant computing platforms. However, then, it faces an inevitable bottleneck, since the time required for partitioning a workflow may delay the next queue of arriving workflows for scheduling. The waiting time may significantly increase when involving a metaheuristic technique–which is known for its compute-intensive pre-processing–in the planning phase. As the size of the workflow increases, this pre-processing time may become longer and produce a larger queue with significant waiting time delay. Hence, we did not consider solutions that schedule each workflow independently, depicted in Fig. 1.3a, since this approach is no different from scheduling a single workflow.

Instead, we considered scheduling algorithms designed to schedule multiple workflows simultaneously, as shown in Fig. 1.3b. Within this context, the addressed multiple workflows scheduling problem focuses on how to allocate the tasks $T = (t_1, t_2, \ldots, t_n)$ from different workflows $W = \{w_1, w_2, w_3, ..., w_n\}$ on a multi-tenant distributed computing systems with several computational resources $R = \{r_1, r_2, r_3, ..., r_n\}$ so that the idle time slots generated from executing a workflow w_i, on a resource r_i, can be allocated to the task t_j from another workflow w_j. There are many benefits and challenges of scheduling multiple workflows using this approach. The main benefits are related to reusing and sharing of the idle time slots and the reduction of waiting time from queueing delay of scheduled workflows. On the other hand, the challenges to achieve these are not trivial. Handling the workloads heterogeneity, managing the continuous arriving workflows, implementing general scheduling approaches that deal with different requirements, and dealing with performance variability in distributed systems are questions that must be answered.

1.3.1 Challenges

In this section, we address several challenges related to the multiple workflows scheduling within the context of the WaaS platform. The problems come from the heterogeneity of workload, which is inflicted by the diversity of users and their specified QoS requirements. Thus, the providers must design the platform that carefully considers the fairness for each user while taking a precaution of uncertainties that may be encountered from the underlying cloud computing infrastructures.

Workload Heterogeneity

A multi-tenant platform that receives multiple requests from various users is expected to face the problems in managing the heterogeneous workload. This workload heterogeneity may take in numerous forms. From the workflow applications perspective, the workload in the WaaS platform may include the different types of scientific applications, which implies the possible existence of software dependencies and library conflicts. The workload may also consist of a variety of workflows' sizes related to the number of nodes, input data needed, and output data generated. How the platform anticipates the heterogeneity of the workflow applications is a big challenge. Furthermore, the heterogeneity may arise from the users' perspective, as each of them must bring up various QoS requirements. In this case, the providers cannot provide one type of service level agreements for all users. They have to tailor the platform to different users' characteristics.

Quality of Service Diversity

Workflow scheduling algorithms are designed to find the optimal configuration of task-resource mapping. However, each user in the WaaS platform may have different QoS requirements. In general, there are two common QoS in scheduling workflows, time and cost. The majority of the scheduling cases require the algorithms to minimize the overall workflow execution time (i.e., makespan). On the other hand, the cost of executing workflows significantly affects the scheduling decisions in utility-based computational infrastructures such as utility grids and cloud computing environments. In this case, the providers also want to minimize their operational costs for executing the workflows.

These two objectives have opposing goals, and a trade-off between them must be considered. This trade-off then drives the various scheduling objectives, which includes minimizing cost while meeting the deadline (i.e., time limit for execution), minimizing makespan while meeting the budget (i.e., cost limit for execution), and a more flexible objective of meeting deadline and budget. In the WaaS platforms, QoS diversity is inevitable due to the different needs of users to execute their workflows. The diversity is not only related to the value of the user-defined QoS, but also rise in the form of different scheduling objectives. The various QoS requirements must be handled in a way that each user's need can be fulfilled without sacrificing others who are served by the same system. This issue implies the challenge of maintaining fairness between users.

Fairness and Priority

Tailoring the service to different users' requirements must be equipped by the algorithm's ability to maintain the fairness of treatment for each user. While it is commonly understood that fairness does not always mean the same, there must be a standard in defining the priority in scheduling workflows to ensure the fulfilment of different SLA without having anyone to be sacrificed. For example, the platform should be able to manage between the users who prefer faster execution time and willing to spend more monetary cost with the ones that have a cost restriction but relaxed time limit. The platform may prioritize the first user and postponed the latter as long as both SLAs are not violated. This kind of priority assignment is crucial, since it will impact the fairness treatment between different users within the platform.

Performance Variability

The virtualization-based cloud computing environments create an inevitable overhead issue. The isolation that hides the underlying hardware in the data center generates overhead performance delays. The multi-tenancy aspect in a cloud computing environment that exploits the virtualization to serve many users is one of the sources of the uncertainties within the cloud's performance variability. Therefore, it is uncommon for cloud instances to exhibit performance degradation at different times when the users' requests are at its highest. Since most of the scheduling relies on the estimate of the tasks' runtime, the performance variability in cloud computing environments creates an issue to be considered when designing algorithms for WaaS platform.

1.4 Thesis Contributions

Based on the research problem definition and its challenges, this thesis makes the following **key contributions**:

- The identification and description of the challenges of the WaaS platform that must be addressed by the scheduling and resource provisioning algorithm;
- A taxonomy based on workload, deployment, priority assignment, task scheduling, and resource provisioning model of state-of-the-art multiple workflow scheduling algorithms in multi-tenant distributed systems;
- A survey and detailed discussion of state-of-the-art approaches in scheduling and resource provisioning algorithms within the thesis' scope;
- A budget distribution strategy to assign tasks' budget based on various types of heterogeneous resources to fit into the specific granularity of cloud instances leasing periods;
- A heuristic algorithm that dynamically schedules tasks driven by their sub-budget while enforcing a resource-sharing policy for multiple workflows, which is capable of minimizing the makespan while meeting the user-defined budget;
- An online and incremental learning approach in resource estimation and runtime prediction for scientific workflows that takes into consideration the temporal aspect of cloud resources performance degradation;

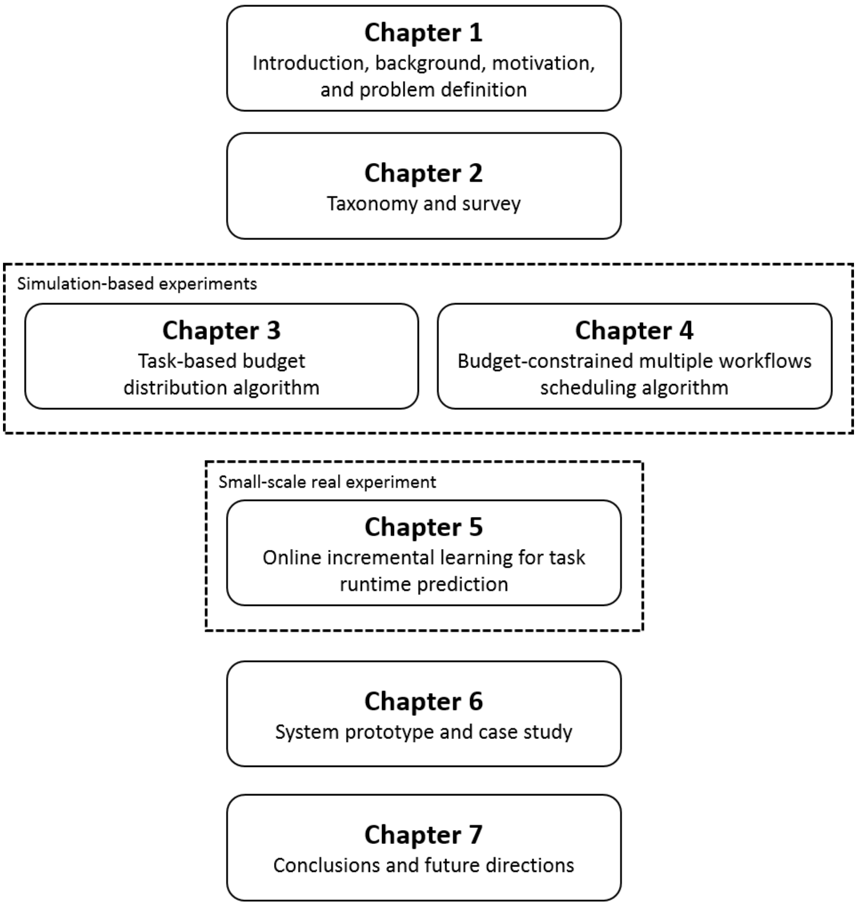

Chapter 1
Introduction, background, motivation, and problem definition
Chapter 2
Taxonomy and survey
Simulation-based experiments
Chapter 3
Task-based budget distribution algorithm
Chapter 4
Budget-constrained multiple workflows scheduling algorithm
Small-scale real experiment
Chapter 5
Online incremental learning for task runtime prediction
Chapter 6
System prototype and case study
Chapter 7
Conclusions and future directions

Fig. 1.4: The core structure of the thesis

- The implementation of multiple workflows scheduling algorithm and their integration to an existing cloud workflow management system for the workflow-as-a-service cloud platform.

1.5 Thesis Organization

The chapters of this thesis are derived from several conference and journal papers produced during the PhD candidature. The core structure of the thesis is depicted in Fig. 1.4 and organized as follows:

- Chapter 1 presents identification and description of challenges to the workflow-as-a-service cloud platform based on the definition of the problems previously discussed.
- Chapter 2 presents a comprehensive taxonomy and survey on multiple workflows scheduling algorithms in multi-tenant distributed systems. This chapter is derived from:

  - **Muhammad H. Hilman**, Maria A. Rodriguez, and Rajkumar Buyya. ‘Multiple Workflows Scheduling in Multi-tenant Distributed Systems: A Taxonomy and Future Directions.‘ *ACM Computing Surveys (CSUR), vol. 53, no. 1*, Pages 10:1-10:39, 2020.

- Chapter 3 presents a task-based budget distribution strategy that takes into consideration the granularity of cloud resources billing periods set by the cloud providers. This chapter is derived from:

  - **Muhammad H. Hilman**, Maria A. Rodriguez, and Rajkumar Buyya. ‘Task-based Budget Distribution Strategies for Scientific Workflows with Coarse-grained Billing Periods in IaaS Clouds.‘ *In Proceedings of the 13th IEEE International Conference on e-Science (e-Science)*, Pages 128-137, 2017.

- Chapter 4 presents a budget-constrained scheduling algorithm that incorporates the resource sharing policy to minimize the makespan, while meeting the budget in multiple workflows scenarios. The algorithm dynamically allocates tasks to the cloud resources driven by its sub-budget, which is distributed based on the strategies introduced in Chapter 3. This chapter is derived from:

  - **Muhammad H. Hilman**, Maria A. Rodriguez, and Rajkumar Buyya. ‘Resource-sharing Policy in Multi-tenant Scientific Workflow-as-a-Service Platform.‘ *Journal of Computer and System Sciences (JCSS)*, 2020 (under review).

- Chapter 5 presents an online and incremental learning approach to resource consumption and runtime prediction of scientific workflows in the WaaS platform, which considers the performance variability of clouds based on their temporal provision and computational capacity (i.e., VM type). This chapter is derived from:

  - **Muhammad H. Hilman**, Maria A. Rodriguez, and Rajkumar Buyya. ‘Task Runtime Prediction in Scientific Workflows Using an Online Incremental Learning Approach.‘ *In Proceedings of the 11th ACM/IEEE International Conference on Utility and Cloud Computing (UCC)*, Pages 93-102, 2018.

- Chapter 6 presents the functionality of existing workflow management systems and its extension to schedule multiple workflows for the workflow-as-a-service cloud platform. A case study is presented using two bioinformatics workflows application, which includes the real-system implementation of approaches discussed in Chapter 4 and 5. This chapter is derived from:

  - **Muhammad H. Hilman**, Maria A. Rodriguez, and Rajkumar Buyya. ‘Workflow-as-a-Service Cloud Platform and Deployment of Bioinformatics Workflow Applications.‘ *Knowledge Management in Development of Data-Intensive Software Systems (KMD-DIS)*, 2020 (under review).

- Chapter 7 concludes the thesis, summarizes its findings, and provides directions for future research.

This page intentionally left blank.

Chapter 2

A Taxonomy and Survey of Multiple Workflows Scheduling Problem

In this chapter, we propose a taxonomy that classifies multiple workflows scheduling algorithms based on their workloads, platform deployment, scheduling, and resource provisioning model. Furthermore, we present a comprehensive survey of multiple workflows scheduling strategies that covers various aspects, including QoS-constrained, energy-aware, and fault-tolerant scheduling solutions. In this way, we display not only state-of-the-art solutions, but also provide future insights and foster the research in the scheduling area for the development of the WaaS platform.

2.1 Introduction

Workflow scheduling was studied and surveyed extensively during the cluster and grid computing era [20] [21]. Subsequently, when cloud computing emerged as a new paradigm with market-oriented focus, the scientific community got a promising deployment platform for workflow applications offering multiple benefits. However, this emerging paradigm also brought forth additional challenges. Solutions for cloud workflow scheduling have been extensively researched, and a variety of algorithms have been developed [22] [23]. Furthermore, various existing taxonomies of workflow scheduling in clouds focus on describing the particular scheduling problem as well as its unique challenges and ways of addressing them [24] [25] [26]. Contrary to these previous studies that focus mostly on a single workflow scheduling, this chapter addressed the scheduling problem from a higher level perspective–it considered the scheduling of multiple workflows that arrive continuously to a WaaS platform.

This chapter is derived from: **Muhammad H. Hilman**, Maria A. Rodriguez, and Rajkumar Buyya. 'Multiple Workflows Scheduling in Multi-tenant Distributed Systems: A Taxonomy and Future Directions.' *ACM Computing Surveys (CSUR), vol. 53, no. 1*, Pages 10:1-10:39, 2020.

In this chapter, various aspects of existing multiple workflows scheduling algorithms are discussed. In particular, we present the workload, deployment, priority assignment, task scheduling, and resource provisioning models. These taxonomy models correspond to the inherent challenges in managing multi-tenant distributed systems which includes the heterogeneity in computational requirements, the various types of underlying infrastructures, the fairness and priority policy, and the trade-off between various QoS requirements in scheduling and resource provisioning phase. Finally, the classification of surveyed algorithms to the taxonomy is presented to provide a comprehensive understanding of the state-of-the-art algorithms in multiple workflows scheduling.

2.2 Taxonomy

The scope of this chapter is limited to the theoretical algorithms developed for multiple workflows scheduling that represent the problem in the WaaS platforms. In this section, we described various challenges of scheduling multiple workflows and their relevancy for each taxonomy classification. This section is limited to the discussion of each taxonomy definition; the classification and references of algorithms are presented later in Section 2.3.18.

2.2.1 Workload Model

Multiple workflows scheduling algorithms are designed to handle workloads with a high level of heterogeneity that represents a multi-tenant characteristic in the platforms. Workload heterogeneity can be described from several aspects, including the continuous arrival of workflows at different times, the various types of workflow applications that differ in computational requirements, the difference in workflow sizes, and the diversity in software libraries and dependencies.

The different arrival time of multiple workflows in the platforms resembles the problem of streaming data processing that deals with continuous incoming tasks to be processed. In contrast with some static single workflow scheduling algorithms that make use of information (e.g., workflow structure, the runtime of tasks, computational requirements) to create a near-optimal schedule plan, the continuous arrival of workflows in WaaS platforms makes this an unsuitable approach. Furthermore, conventional techniques to achieve near-optimal schedules such as metaheuristics are computationally intensive, and the complexity will grow as the workflow size increases. The time for planning may take longer than the actual workflow execution. Hence, a lightweight dynamic scheduling approach is the most suitable for WaaS platforms, since the algorithms must be able to deal with the dynamicity of the workload. For instance, at peak time, the concurrency of requests may be very high, whereas, at other times, the submission rate may reduce to a point where the inter-arrival time between workflows is long enough to execute each workflow in a dedicated set of resources.

The variety of application types is another issue to be addressed. A study by Juve et al. [27] shows a variety of workflow applications with different characteristics. The Montage astronomy workflow [28] that is used to reconstruct mosaics of the sky is considered as a data-intensive application with high I/O activities. The Cybershake workflow [29] that is used to characterize earthquake hazards using the Probabilistic Seismic Hazard Analysis (PSHA) techniques is categorized as a compute-intensive workflow with multiple reads on the same input data. The Broadband workflow that is used to integrate a collection of simulation codes and calculations for earthquake engineers has a relative balance of CPU and I/O activities in its tasks. These three samples show different types of workflow applications that may have different strategies for scheduling to be carried out. For example, an approach of clustering tasks with a high dependency of input/output data (i.e., data-intensive) and allocating them on the same resource to minimize data transfer.

Furthermore, heterogeneity is also related to the size of the workflows. The size represents the number of tasks in a workflow and may differ even between instances of the same type of workflow application due to different input datasets. For example, the Montage workflow takes the parameters of width and height degree of a mosaic of the sky as input. The higher the degree, the larger the Montage workflow size to be executed as it resembles the size and shape of the sky area to be covered and the sensitivity of the mosaic to produce. A large-scale workflow may raise another issue in scheduling such as high volume data transfer that may cause a bottleneck in the network, which will affect other smaller scale workflows being executed in the platform.

Another heterogeneity issue is the various software libraries and dependencies required for different workflow applications. This problem is related to the deployment and configuration of workflow applications in the systems. Deploying different software libraries and dependencies requires technical efforts in installing the software and managing conflicts between software dependencies. The most important implication related to this case is the resource sharing between workflows to utilize idle time slots produced during the scheduling. In cluster and grid environments where every user uses shared installed software systems on a physical machine, the

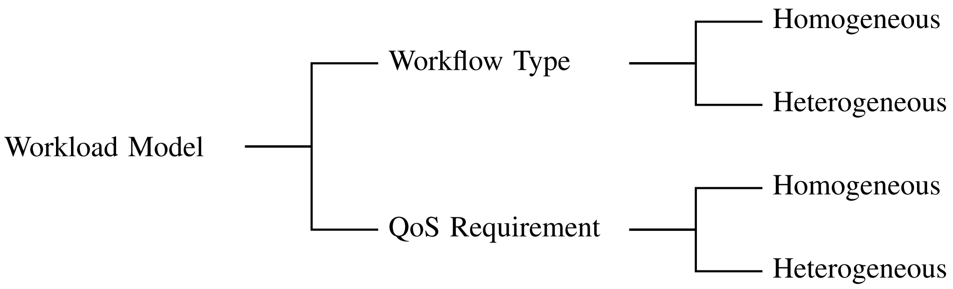


Fig. 2.1: Workload model taxonomy

conflicting dependencies are inevitable. This problem can be avoided by isolating applications in virtualized environments (i.e., cloud computing environments).

However, in clouds where the workflow's deployment and configuration can be isolated in a virtual machine, the possibility to share the computational power between users in a particular virtual machine is limited. This problem is due to the virtual machine capacity (i.e., memory, storage) limitation and possible conflicting dependencies if we want to have as much as software configured in a virtual machine. The trade-off between the isolation and the resource sharing in clouds can be solved using containers as successfully implemented using Kubernetes on Docker containers [30], Singularity and CVMFS at OSG [31] and Shifter at Blue Waters [32]. In this case, container, a lightweight operating system-level virtualization, is used to isolate the workflow application before deploying them on virtual machines. Therefore, both isolation and resource sharing objectives can be achieved. Based on the heterogeneity issue, these workloads can be differentiated by their workflow type and user-defined QoS requirements, as shown in Fig. 2.1.

Workflow Type

Scheduling algorithms for WaaS platforms must consider the fact that the users in this system may submit a single type or different workflow applications. These variations can be categorized into homogeneous and heterogeneous workflow types. A homogeneous workflow type assumes all users submit the same kind of workflow applications (e.g., multi-tenant platforms for Montage workflow). In this case, the algorithms can be tailored to handle a specific workflow application by exploiting its characteristics (e.g., topological structure, computational requirements, software dependencies, and libraries). For example, related to a topological structure, a workflow with a large number of tasks in a level may raise an issue of data transfer. This issue can potentially become a communication bottleneck when all of the tasks in a level concurrently transfer the data input needed to execute the tasks. Therefore, clustering the tasks may result in a reduction in data transfer and eliminates the bottleneck in the system.

Furthermore, the heterogeneity from the resource management perspective affects how the scheduling algorithms handle software dependencies and libraries installed in computational resources. The algorithms for a homogeneous workflow type can safely assume that all resources contain the same software for a workflow application. In this way, the constraints for choosing appropriate resources for particular tasks related to the software dependencies can be eliminated since all of the resources are installed and configured for the same workflow application.

On the other hand, to handle a heterogeneous workflow type, the algorithms must be able to tackle all various possibilities of workflow type submitted into the platforms. In a WaaS platform, where the heterogeneous workflow type is considered, tailoring the algorithms to the specific workflow application characteristics is impractical. The scheduling algorithms must be designed using a more general approach. For example, related to the topological structure, a workflow's task is considered ready for the execution when all of its predecessors are executed, and its data input is available in a resource allocated for execution. In this way, the algorithms can exploit a heuristic to build a scheduling queue by throwing in all tasks with this specification.

Therefore, a variety of software dependencies required for different workflow applications increases the possible conflict of software dependencies in platforms that consider heterogeneous workflow type. In this case, the algorithms must include some rules in the resource selection step to determine what relevant resources can be allocated for specific tasks. For example, the algorithms can define a rule that is only allowing a task to be assigned a resource based on its software dependencies and libraries' availability.

QoS Requirements

Workloads in WaaS platforms must be able to accommodate multiple users' requirements. These requirements are represented by the QoS parameters defined when users submit their workflows to the platforms. We categorized the workloads based on the users' QoS requirements into homogeneous and heterogeneous QoS requirements.

The majority of algorithms designed for the platforms surveyed in this study consider a homogeneous QoS requirement. They are designed to achieve the same scheduling objective (e.g., minimizing the makespan, meeting the budget) for all workflows. Meanwhile, a heterogeneous QoS requirement is addressed by the system's ability to be aware of various objectives and QoS parameters demanded by a particular user. The algorithms may consider several strategies to handle workflows with different QoS requirements. For example, to process workflows that are submitted with the deadline constraints, the algorithms may exploit the option to schedule them into the cheapest resources to minimize operational cost as long as their deadlines can be met. At the same time, the algorithms can also handle workflows with the budget constraints by using another option to lease as much as possible the fastest resources within the available budget.

2.2.2 Deployment Model

Handling the performance variability in multi-tenant distributed computing systems is essentials to the multiple workflows scheduling problems as the scheduling highly relies on the accurate estimation of workflow‘s performance on a particular computational infrastructure. Attempts to increase the quality of scheduling by accurately estimating the time needed for completing a task, as one of the strategies for taking care of the uncertainty has been extensively studied [33]. Specific work designed for scientific workflows includes a work by Nadeem and Fahringer [34] that utilized the template to predict the scientific workflow application execution time. Another work by da Silva et al. [35] introduced an online approach to estimate the resource consumption for scientific workflows. Meanwhile, Pham et al. [36] worked on machine learning techniques to predict task runtime in workflows using a two-stage prediction approach.

When we specifically discuss cloud environments, the uncertainty becomes higher than cluster and grid environments. The virtualization of clouds is the primary source of the performance variability, as reported by Leitner and Cito [4] and also previously discussed by Jackson et al. [37]. The cloud instances performance varies over time due to several aspects, including the virtualization overhead, the geographical location of the data center, and especially the multi-tenancy of clouds. For example, it is not uncommon for a task to have a longer execution time during a specific time in cloud instances (i.e., peak hours) due to the number of users served by a particular cloud provider at that time. The main conclusions from their works substantiate our assumption that the predictability of clouds is something that is not easy to address.

Another variable of uncertainty in clouds is the provisioning and de-provisioning delays of VMs. When a user requests to launch a cloud's instance, there is a delay between the request

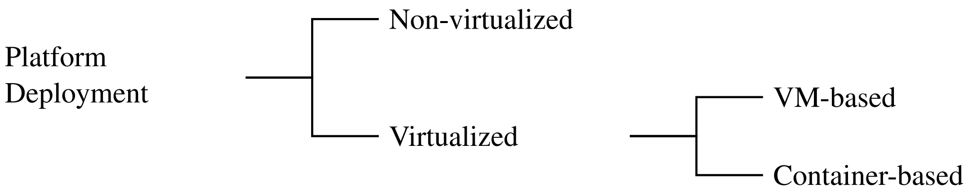


Fig. 2.2: Deployment model taxonomy

and when the VM is ready to use, which is called as provisioning delay. There also exists a delay in releasing the resource, called as de-provisioning delay. Not considering the provisioning and de-provisioning delays in the scheduling phase may cause a miscalculation of when to acquire and to release the VM. This error may cause an overcharging bill of the cloud services. A study by Mao and Humphrey [38] reported that the average provisioning delay of a VM, observed from three cloud providers–Amazon EC2, Windows Azure, and Rackspace–was 97 seconds while more recently, Jones et al. [39] presented a study which shows that three different cloud management frameworks–OpenStack, OpenNebula, and Eucalyptus–produced VM provisioning delays between 12 to 120 seconds. However, delays are not only derived from acquiring and releasing instances. Since most of the WMS treat cloud instances (i.e., virtual machines) as virtual clusters using third-party tools (e.g., HTCondor[1]), there exists a delay in integrating a provisioned VM from cloud providers into a virtual cluster. An upper bound delay of 60 seconds for this process was observed by Murphy et al. [40] for an HTCondor virtual cluster. These delays are one of the sources of uncertainty in clouds, and therefore, the algorithms should consider then to get an accurate scheduling result.

Hence, the scheduling algorithms for WaaS platforms can be differentiated based on its deployment model. We identified two types of algorithms based on their deployment model, as illustrated in Fig. 2.2. Several issues and challenges that arise from these models are worthy of being considered by the scheduling algorithms.

Non-virtualized

The majority of works in our survey design are scheduling algorithms for cluster and grid environments. These two environments are the traditional way of establishing multi-tenant distributed computing systems where a large number of computational resources are connected through a fast

[1] https://research.cs.wisc.edu/htcondor/

network connection so that many users in a shared fashion can utilize the infrastructure. However, in this way, there is no isolation between software installed related to their dependencies and libraries within the same physical machine.

Accommodating multi-tenant users in a non-virtualized environment is limited by the computational infrastructure static capacity. This staticity makes it very hard to auto-scale the resources in non-virtualized environments. Thus, the algorithms cannot efficiently serve a dynamic workload without having a queueing mechanism to schedule overloaded requests at a particular time. For example, adding a node into an established cluster infrastructure is possible but may involve technicalities that cannot be addressed in a short period. This environment also does not allow the users to shrink and expand their allocated resources quickly since the changes need to go through the administrator intermediaries. Therefore, the primary concern of scheduling algorithms designed for this environment is to ensure for maximum utilization of available resources, so the algorithms can reduce the queue of users waiting to execute their workflows. In this case, techniques such as task rearrangement [41] and backfilling [42] can be used to fill the gaps produced by scheduling a particular workflow, by allocating these idle slots to other workflows.

Virtualized

The algorithms designed for virtualized environments (i.e., clouds) can gain advantages from a flexible configuration of VM as it isolates specific software requirements needed by a user in a virtualized instance. A fully configured virtual machine can be used as a template and can be shared between multiple users to run the same workflows. This isolation ensures little disturbance to the platforms and the other users whenever a failure occurs. However, in this way, the possible computational sharing of a virtual machine is limited. It is not plausible to configure a virtual machine for several workflow applications at the same time. In this case, containers can be used to increase configuration flexibility in virtualized environments. The container is an operating-system-level virtualization method to run multiple isolated processes on a host using a single kernel. The container is initially a feature built for Linux that is further developed and branded as a stand-alone technology (e.g., Docker[2]) that not only it can run on Unix kernel, but also on Windows NT kernel (e.g., windows container[3]). A full workflow configuration can be created in a

[2] https://www.docker.com/

[3] https://docs.microsoft.com/en-us/virtualization/windowscontainers/

container before deploying it on top of virtual machines. In this way, the computational capacity of VMs can be shared between users with different workflows.

In the context of scalability, algorithms designed for virtualized environments can comfortably accommodate multi-tenant requirements. The algorithms can acquire more resources in on-demand fashion whenever requests are overloading the system. Furthermore, this on-demand flexibility supported by the pay-as-you-go pricing scheme reduces the burden for the WaaS platform providers to make upfront payments for reserving a large number of resources that may only be needed at a specific time (i.e., peak hours). Even if a particular cloud provider cannot meet the demand of the WaaS platform providers, the algorithms can provision resources from different cloud providers.

However, this environment comes with a virtualization overhead that implies a significant performance variability. The overhead not only occurs from the communication bottleneck when a large number of users deal with high volume data but also the possible degradation of CPU performance since the computational capacity is shared between several virtual machines in the form of virtual CPU. The other overheads are the delay in provisioning and de-provisioning virtual machines and the delay in initiating and deploying the container. The scheduling algorithms have to deal with these delays and consider them in the scheduling to ensure accurate results.

2.2.3 Priority Assignment Model

Fairness and priority issues are unavoidable in multiple workflows scheduling. Given two workflows that arrive at the same time, the decision to execute a particular workflow must be determined using a policy to ensure that limited computational resources can be fairly allocated. This fairness can be identified from the *slowdown*, a difference of expected makespan between the execution of a workflow in dedicated resources vs resource sharing environments [43]. In this case, the scheduling algorithms are designed to gain a slowdown value as low as possible. Since the computational resources are limited, the slowdown is inevitable. Therefore, the algorithms assigned a priority to the workflows to ensure the fulfilment of QoS. The priority can be derived from several aspects, including the QoS defined by users, the type of workflow application, the user's preference, and the size of the workflows.

Priority assignment can be determined based on the user-defined QoS. It is evident for scheduling algorithms to prioritize workflow with the earliest deadline, as this can ensure the fulfilment

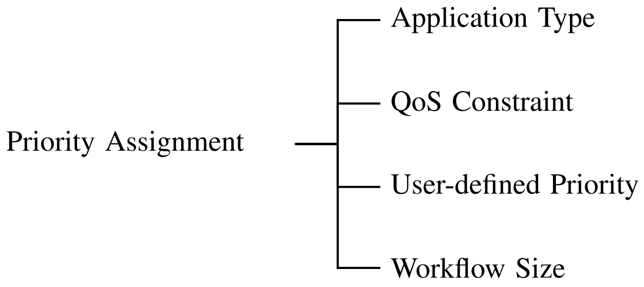


Fig. 2.3: Priority assignment model taxonomy

of QoS requirements. In this way, algorithms may introduce a policy based on the deadline that delays the scheduling of a workflow with a more relaxed deadline to improve resource sharing in the system without violating the fairness aspect. On the other hand, the priority assignment can be defined from the budget. In real-world practice, it is common to prioritize the users with more budget to do a particular job compared to the lower one (e.g., priority check-in for business class passengers). This policy also can be implemented in multiple workflows scheduling.

Assigning priority based on the type of application can be done by defining application or user classes. For example, workflows submitted for education or tutorial purpose may have a lower priority than the workflows executed in a scientific research project. Meanwhile, a workflow that is used to predict the typhoon occurrence may be performed first compared to a workflow for creating the mosaic of the sky. This policy can be defined out of the scheduling process based on some strategies adopted by the providers.

Moreover, the priority assignment can also be determined based on the size of workflows. This approach is the most traditional way of priority scheduling that has been widely implemented, such as the Shortest Job First (SJF) policy, which prioritizes the smaller workflows over a larger one to avoid starvation. Another traditional scheduling algorithm like Round-Robin (RR) also can be constructed based on the size of the workflows to ensure both of the small and large workflows get fair treatment in the systems.

Fairness between users in WaaS platforms can be achieved through priority assignments. This assignment is crucial as the ultimate goal of the providers is to fulfil users' QoS requirements. We identified various priority assignment models from surveyed algorithms that consider the type of workflow application, users QoS constraints, user-defined priority, and size of workflows in their design, as shown in Fig. 2.3.

Application Type

Different types of workflow applications can be used to define the scheduling priority based on their context and critical functionality. The same workflow application can differ in priority when it is used in a different environment. Montage workflow used for an educational purpose may have a lower priority than a solar research project using the same workflow. Meanwhile, considering the different critical functions of workflows and some events (e.g., earthquake), some workflow applications can be prioritized from the others. For example, Cybershake workflow to predict the ground motion after an earthquake may be prioritized compared to the Montage workflow that is used to create a mosaic of the sky image. This priority assignment needs to be designed in a specific policy of the providers that can regulate the fairness of the scheduling process.

QoS Constraint

Deriving priority assignments from users' QoS constraints can be done within the scheduling algorithms. This assignment is included in the logic of algorithms to achieve the scheduling objectives. For example, an algorithm that aims to minimize cost while meeting the deadline may consider to de-prioritize and delay the task of a particular workflow that has a more relaxed deadline to re-use the cheapest resources available. In this way, the algorithms must be designed to be aware of the QoS constraints of the tasks to derive these parameters into a priority assignment process.

Furthermore, the challenge of deriving priority assignment from QoS constraints may come from a heterogeneous QoS requirement workload. The algorithms must be able to determine a priority assignment for multiple workflows with different QoS requirements. For example, given two workflows with different QoS parameters, a workflow was submitted with a deadline, while another was included with a budget. The priority assignment can be done by combining these constraints with its application type, user-defined priority, or workflow structure.

User-defined Priority

On the contrary to the application type priority model that may be arranged through a specific policy, the priority assignment must also consider the user-defined priority in scheduling algorithms. This priority can be defined by users with appropriate compensations for the providers. For example, it is not uncommon in real-world practice to spend more money to get a prioritized treatment that affects the speed of process and quality of service (e.g., regular and express postal service).

It is possible in WaaS platforms to accommodate such a mechanism where the users are given the option to negotiate their priority through a monetary cost compensation for the providers. This mechanism is a standard business practice adopted in WaaS platforms (e.g., reserved, on-demand, and spot instances pricing schemes).

Workflow Size

Another approach on priority assignment is based on the structure of workflows (e.g., size, parallel tasks, critical path). Prioritizing workflows based on their sizes resembles a traditional way of priority scheduling, such as Shortest Job First (SJF) policy that gives priority to the shortest tasks, and Round Robin (RR) policy that attempts to balance the fairness between tasks with different sizes. This prioritization can be combined with the QoS constraint to produce better fairness. For example, a large-scale workflow may have an extended deadline. Therefore, smaller workflows with tight deadlines can be scheduled between the tasks' execution of this larger workflow.

2.2.4 Task Scheduling Model

Task-resource mapping is the primary activity of scheduling. All of the workflow scheduling algorithms have the purpose of finding the most optimal configuration of task-resource mapping. However, each scheduling problems may have different requirements regarding the QoS. In general, there are two standard QoS requirements in workflow scheduling, time and cost. Most of the cases require the algorithms to minimize the overall execution time (i.e., makespan).

On the other hand, the cost of executing the workflows significantly affects the scheduling decisions in utility-based computational infrastructures such as utility grids and cloud environments. Every user wants to minimize the cost of executing their workflows. These two objectives have opposing goals, and a trade-off between them must be considered. This trade-off then is derived into various scheduling objectives such as minimizing cost while meeting the deadline (i.e., time limit for execution), minimizing makespan while meeting the budget (i.e., cost limit for execution), or a more flexible objective, meeting deadline, and budget.

In WaaS platforms, QoS diversity is inevitable due to the different users' needs in executing their workflows. The variety is not only related to the QoS values the users define but also raise in the form of different scheduling objectives. The various user-defined QoS requirements must be handled in a way that each user's need can be fulfilled without sacrificing the other users.

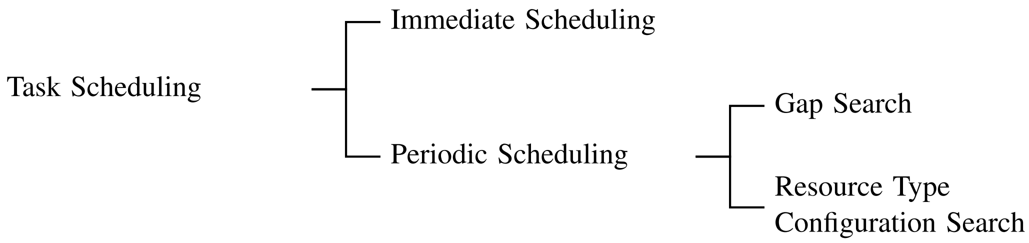


Fig. 2.4: Task scheduling model taxonomy

All algorithms in this study avoid metaheuristics approaches that are known for their complexity in planning the schedule before runtime. This planning creates an overhead waiting delay as the continuous arriving workflows have to wait for pre-processing before the actual scheduling takes place. Therefore, they use dynamic approaches that reduce the need for intensive computing at the planning phase and aim to achieve a fast scheduling decision by considering the current status of the systems. These approaches can be divided into immediate and periodic scheduling, as illustrated in Fig. 2.4.

Immediate Scheduling

Immediate scheduling or just-in-time scheduling is a dynamic scheduling approach in which tasks are scheduled whenever they are ready for scheduling. In the case of multiple workflows, this scheduling approach collects all of the ready tasks from different workflows in a task pool before deciding to schedule based on some particular rules. The immediate scheduling tries to overcome the fast dynamic changes in the environments by adapting the decision based on the current status of the system. However, as the algorithm schedules the tasks based on a limited amount of information (i.e., a limited view of the previous and future information), this approach cannot achieve an optimal scheduling plan. On the other hand, it is an efficient way for multi-tenant platforms that deal with uncertain and dynamic environments.

The immediate scheduling resembles list-based heuristics scheduling. This scheduling approach, in general, has three scheduling phases, task prioritization, task selection, and resource selection. The algorithms repeatedly select a particular task from the scheduling queue that is constructed based on some prioritization method and then pick the appropriate resource for that specific task. For example, in deadline constraint-based heuristics algorithms that aim to minimize the cost while meeting the deadline, the scheduling queue is constructed based on the earliest deadline first (EDF) of the tasks, and the cheapest resources that can meet the deadline are chosen to minimize the cost. The time complexity for heuristic algorithms is low. Therefore, it is suitable for multiple workflows scheduling algorithms that deal with the speed to manage the scheduling.

Periodic Scheduling

This approach schedules the tasks periodically to exploit the possibility of optimizing a set of tasks' scheduling within a period. While in a general batch scheduling, a particular set is constructed based on the size of workload (i.e., schedule the tasks after reaching a certain number), the periodic approach schedules the tasks in a set of timeframe. In this case, the periodic scheduling acts as a hybrid approach between static and dynamic scheduling methods. Static means that, the algorithms exploit the information of a set of tasks (i.e., structures, estimated runtime) to create an optimal plan, but it does not need to wait for a full workload of tasks to be available. The dynamic sense of algorithms, however, adapts and changes the schedule plan periodically. Hence, periodic scheduling refers to a scheduling technique that utilizes the schedule plan of a set of tasks available in a certain period to produce a better scheduling result. This method is more adaptable to changes and has faster pre-runtime computation than static scheduling techniques since it only includes a small fraction of workload to be optimized rather than the entire workload. On the other hand, this approach can achieve a better result from having an optimized schedule plan than typical immediate scheduling but with less speed in scheduling.

One of the approaches in periodic scheduling is identifying the gaps between tasks during the schedule. The identification uses an estimated runtime of tasks and their possible position in a resource during runtime. The most common methods to fill the gap are task rearrangement and backfilling strategy. Task arrangement strategy re-arranges the tasks scheduling plan to ensure the minimal gap in a schedule plan. At the same time, backfilling allocates the ordering list of tasks and then backfills the holes produced between the allocation using the appropriate tasks. Both strategies do not involve an optimization algorithm that requires intensive computation since the WaaS platforms consider speed in the schedule to cope with the users' QoS requirements.

While gap search is related to the strategy for improving resource utilization, another approach utilizes resource type configuration to optimize the cost spent on leasing computational resources. In a heterogeneous environment where resources are leased from third-party providers with some monetary costs (i.e., utility grids, clouds), determining resource type configuration to optimize

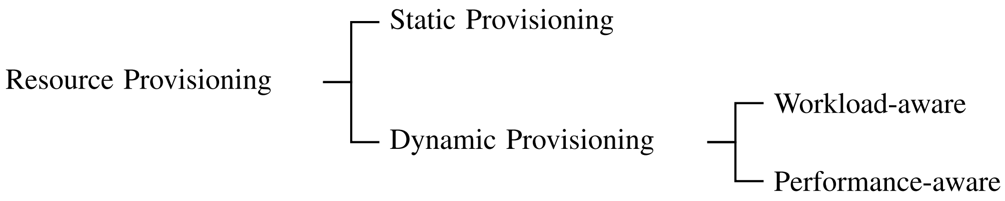


Fig. 2.5: Resource provisioning model taxonomy

the cost of leasing resources is necessary. For example, Dyna algorithm [44] that considers the combination use of on-demand and spot instances in Amazon EC2, utilizes heuristics to find the optimal resource type configuration to minimize the cost.

2.2.5 Resource Provisioning Model

Resource provisioning forms an essential pair with task scheduling. In this stage, scheduling algorithms acquire and allocate resources to execute the scheduled tasks. We derived the categorization of resource provisioning based on the ability of scheduling algorithms to expand and shrink the number of resources within the platforms to accommodate the dynamic workloads of the WaaS platforms, as shown in Fig. 2.5.

Static Provisioning

The static provisioning refers to scheduling algorithms where the number of resources used is relatively constant along the scheduling process. Therefore, the primary issue is related to the algorithms' ability to optimize the available resources to accommodate multiple users. This condition can be observed from the algorithms that emphasize heavily on the prioritization technique for workflows to be scheduled due to the limited available resources contested by many users. Another aspect is the improvement of resource utilization of the systems, which describes the ability of algorithms to allocate a limited number of resources efficiently.

This static provisioning is not exclusive to the non-virtualized environment (e.g., clusters, grids), where the number of resources is hardly changing over time. This case also prevails in cloud environments where the providers determine the number of VMs to be leased before initiating the platforms, and the number remains unchanged over time. In this scenario, the scheduling algorithms do not consider any resource provisioning strategy to scale up and down resources when facing a dynamic workload of workflows.

Dynamic Provisioning

As the clouds provide elastic provisioning of virtual machines, the scheduling algorithms of WaaS platforms in clouds take advantage of the dynamic provisioning approach. The automated scaling of resources that can be easily implemented in clouds has been widely adopted. Specifically, the scheduling algorithms that deal with a dynamic workload, where the need for resources can be high at a point (i.e., peak hours), while at the same time, the operational cost must be kept at the minimum. To minimize the operational cost, the leased VMs have to be released when the request is low. From the existing algorithms, at least, there are two different approaches to auto-scale the cloud instances, workload-aware and performance-aware.

Workload-aware dynamic provisioning is related to the ability of algorithms to become aware of the workload status in WaaS platforms, and then to act according to the situation. For example, the platforms are acquiring more VMs to accommodate the peak condition. One of the heuristics used in this scenario is based on the deadline constraints of the workload. For example, the algorithms use a task's deadline to decide whether a task should re-use available VMs, provision a new VM that can finish before the deadline, or delay the schedule to re-use future available VMs as long as it does not violate the deadline. This decision is essential, since the dynamic workload is typical in WaaS platforms, where the systems cannot predict the future status of the workload. Using this heuristic provisioning, additional VMs are more accurate as the acquired new VM is based on the requirement of a particular task being scheduled.

On the other hand, the performance-aware dynamic provisioning refers to an approach of VMs auto-scaling based on total resource utilization of current provisioned VMs. The algorithms monitor the system's status and acquire additional VMs when the usage is high and release several idle VMs when the utilization is low. Maintaining resource utilization at a threshold ensures the efficiency of WaaS platforms in the scheduling process. The majority of works considering this approach are the ones that consider only homogeneous VM types in their systems. In this way, the algorithms do not need to perform the complicated selection process of the VM types.

2.3 Survey

In this section, we discuss a number of surveyed multiple workflows scheduling algorithms published between 2008 to 2019 that are relevant to our scope.

Table. 2.1: Experimental design of RANK_HYBD algorithm

Experiment Type	Simulation	
Workload	Synthetic workflows	: Taken from Hönig et al. [47]
	Number of workflows	: 25 workflows
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (175–249)	: Number of tasks
	Meshing degree	: Nodes connectivity
	Edge-length	: Average number of nodes between two connected nodes
	CCR	: Ratio of computation and communication time
Performance Metrics	Makespan	: Total execution time
	Turnaround time	: Makespan and waiting time
	Resource utilization	: Percentage of time when computational resources are busy

2.3.1 Planner-Guided Scheduling for Multiple Workflows

RANK_HYBD algorithm [45] was introduced to overcome the impracticality of the ensemble approach (i.e., merging multiple workflows) to handle different submission time of workflows to the system by scheduling individual tasks dynamically. The algorithm put together all ready tasks from different workflows into a pool. Then, the algorithm used a modified upward ranking [46], which calculated the weight of a task based on its relative position from the exit tasks and estimated computational length to assign individual tasks priorities. This task ranking time complexity is $\mathcal{O}(T_w.P_s)$ for all tasks in a workflow T_w given a set of static processors P_s. In contrast with the original upward ranking implementation in HEFT algorithm that chooses tasks with higher rank value, RANK_HYBD prefers tasks with the lowest rank in the pool which creates a time complexity of $\mathcal{O}(T_r.P_s)$ for re-prioritizing all ready tasks T_r. In this case, HEFT prefers the tasks from later arriving workflows and the tasks with the most extended estimated runtime, which creates an unfair pre-emptive policy for the running workflows. By using the opposite approach, the RANK_HYBD algorithm avoids the pre-emptive scheduling delay of a nearly finished workflow if a new workflow is submitted in the middle of the execution. Finally, it schedules each task to the processor that can give the earliest processing time with $\mathcal{O}(T_r(T_r.P_s))$ time complexity. The detailed experimental design for RANK_HYBD algorithm is depicted in Table. 2.1.

In general, the time complexity is quadratic to the number of tasks. The report shows that RANK_HYBD outperformed RANDOM and FIFO algorithms by 1.77x average speedup on workloads up to 25 multiple workflows. This algorithm is the first solution for multiple workflows scheduling. Many later algorithms have adopted the approach to tackle the dynamic workload of workflows using dynamic prioritization for tasks within a workflow and between workflows. Although many aspects, such as QoS constraints, performance variability, and real workflow applications, have not been included in the experiment, this work becomes an important benchmark.

2.3.2 Multiple QoS Constrained Scheduling for Multiple Workflows

Multiple QoS Constrained Scheduling Strategy of Multi-Workflows (MQMW) algorithm [48] incorporated a similar strategy of RANK_HYBD to schedule multiple workflows. MQMW prioritized tasks dynamically based on several parameters, including resource requirement of a task, time and cost variables, and covariance value between time and cost constraint. This task ranking time complexity is $\mathcal{O}(T_w.C_s)$ for all tasks in a workflow T_w given a set of static cloud instances C_s. The algorithm preferred the tasks with a minimum requirement of resources to execute, minimum time and cost limit, and a task with minimum covariance between its time and cost limit (i.e., when time limit decreases, the cost will increase). Each time the scheduling takes place, MQMW re-compute all ready tasks T_r by $\mathcal{O}(T_r.C_s)$ time complexity. Finally, MQMW schedules each task to the best fit idle cloud instances with $\mathcal{O}(T_r(T_r.C_s))$ time complexity.

In general, the time complexity is quadratic to the number of tasks processed. It is tested against RANK_HYBD, even though the RANK_HYBD did not consider the cost in the scheduling constraint. The evaluation results show that MQMW outperformed the success rate of RANK_HYBD [45] algorithm by 22.7%. MQMW is the first attempt to provide the solution of multiple workflows scheduling on the cloud computing environment. However, their cloud model does not resemble the real characteristics that are inherent in clouds such as elastic scalability of instances, on-demand resources, pay-as-you-go pricing schemes, and performance variability of cloud computing environments.

MQSS algorithm [49] was proposed to overcome shortcomings from the MQMW algorithm, which considered additional QoS in the scheduling and the adoption of a more optimal scheduling strategy. With the relatively same approaches, the MQSS includes other QoS parameters into the scheduling's attributes (e.g., time, cost, availability, reputation, and data quality). In general, MQSS has the same time complexity as MQMW, with 12.47% success rate improvement for the same workloads. The detailed experimental design for MQMW and MQSS algorithms is depicted in Table. 2.2 and Table. 2.3 respectively.

Table. 2.2: Experimental design of MQMW algorithm

Experiment Type	Simulation	
Workload	Synthetic workflows	: Randomly generated
	Number of workflows	: 25 workflows
Performance Metrics	Makespan	: Total execution time
	Cost	: Cloud resource monetary cost
	Success rate	: Percentage of successfully executed workflows

Table. 2.3: Experimental design of MQSS algorithm

Experiment Type	Simulation	
Workload	Synthetic workflows	: Randomly generated
	Number of workflows	: 30 workflows
Performance Metrics	Relative weight of resource	: Different computational capacity of resources
	Success rate	: Percentage of successfully executed workflows

2.3.3 Fairness in Multiple Workflows Scheduling

Parallel Task HEFT (P-HEFT) algorithm [50] was the first work of a group from the Universidade do Porto that modelled the non-monotonic tasks (i.e., the execution time of a task might differ on the different number of resources). The algorithm used a relative length position of a task from the entry task (i.e., top-level) and exit task (i.e., bottom-level) to assign the priorities between tasks. This ranking time complexity is $\mathcal{O}(T_w.P_s)$ for all tasks in a workflow T_w given a set of static processors P_s. In their case, the task model is different from other works as it allows parallel execution of a task in several processors. Furthermore, the processor selection and task scheduling for all ready tasks T_r is $\mathcal{O}(T_r(T_r.P_s))$. In general, the complexity is quadratic to the number of tasks processed. The evaluation results show that P-HEFT outperformed static algorithm HPTS [51] by 1.52x average speedup on workloads of 12 multiple workflows. The detailed experimental design for P-HEFT algorithm is depicted in Table. 2.4.

The next work from this group was the Fairness Dynamic Workflow Scheduling (FDWS) algorithm [43]. FDWS chose a single ready task from each workflow into the pool instead of putting all ready tasks together. Local prioritization within a workflow utilized upward-rank mechanism. Furthermore, the task selection from different workflows was based on a percentage of remaining task number of workflow the task belongs to (PRT) and a task position in its workflow's critical

Table. 2.4: Experimental design of P-HEFT algorithm

Experiment Type	Simulation	
Grid Settings	Number of processors	: 50 processors
	Maximum processor speed	: 400 Mflops/s
	Network bandwidth	: 100 Mbps
	Network latency	: 50 μs
Workload	Synthetic workflows	: Randomly generated
	Number of workflows	: 12 workflows
	Arrival intervals	: 40% elapsed time of the last job
Workflow Properties	Width	: Number of task on the largest level
	Regularity	: Uniformity of the number of task in each level
	Density	: Number of edges between two levels
	Jumps	: Edge connection between two consecutive levels
Performance Metrics	Makespan	: Total execution time
	Efficiency	: Ratio of sequential execution and parallel time

Table. 2.5: Experimental design of FDWS algorithm

Experiment Type	Simulation using SimGrid [53]	
Grid Settings	Number of processors	: 280 processors
	Maximum processor speed	: 13–30 Gflops/s
	Platforms derivation	: Based on Grid5000 deployed in France [54]
Workload	Synthetic workflows	: Randomly generated
	Number of workflows	: 50 workflows
	Arrival intervals	: 0–90% elapsed time of the last job
Workflow Properties	Width (20–50)	: Number of task on the largest level
	Regularity (0.2–0.8)	: Uniformity of the number of task in each level
	Density (0.2–0.8)	: Number of edges between two levels
	Jumps (1–3)	: Edge connection between two consecutive levels
Performance Metrics	Makespan	: Total execution time
	Turnaround time	: Total execution and waiting time
	Turnaround time ratio	: Ratio of turnaround time and the minimum makespan
	Normalized turnaround time	: Ratio of minimum and actual turnaround time
	Win	: Percentage of a workflow got the shortest makespan

path (CPL). This prioritization time complexity is $\mathscr{O}(T_w.P_s)$ for all tasks in a workflow T_w given a set of static processors P_s. Meanwhile, resource selection and task scheduling for all ready tasks T_r in FDWS is $\mathscr{O}(T_r(T_r.P_s))$. In general, the complexity is quadratic to the number of tasks processed. The evaluation results show that FDWS outperformed RANK_HYBD [45] and OWM [52] by 1.1x and 1.15x average speedup respectively on workloads of 50 multiple workflows. The detailed experimental design for FDWS algorithm is depicted in Table. 2.5.

Table. 2.6: Experimental design of MW-DBS algorithm

Experiment Type	Simulation using SimGrid [53]	
Grid Settings	Number of processors	: 20–64 processors
	Platforms derivation	: Based on Grid5000 deployed in France [54]
Workload	Synthetic workflows	: Randomly generated based on Juve et al. [27]
	Number of workflows	: 50 workflows
	Arrival intervals	: 10–50% elapsed time of the last job
Workflow Properties	Node (30–100)	: Number of task on the largest level
	Regularity (0.2–0.5)	: Uniformity of the number of task in each level
	Density (0.2–0.5)	: Number of edges between two levels
	Jumps (1–4)	: Edge connection between two consecutive levels
	Deadline (1–2)	: Based on min. and max. execution time
	Budget (0–1)	: Based on min. and max. execution cost
Performance Metrics	Planning successful rate	: Ratio of successfully planed and executed workflows

Multi-Workflow Deadline-Budget Scheduling (MW-DBS) algorithm [55] was their work that addressed the utility aspect of a heterogeneous multi-tenant distributed system. This algorithm includes deadline and budget as constraints. Furthermore, local priority is assigned using the same method from FDWS which creates $\mathcal{O}(T_w.P_s)$ time complexity. However, instead of using PRT and CPL, MW-DBS uses the task's deadline and workflow's scheduled tasks ratio for assigning global priority. Finally, MW-DBS modifies the processor selection phase, in which it includes a budget limit for task processing as a quality measure with $\mathcal{O}(T_r.P_s)$ time complexity. Furthermore, the complexity of resource selection and task scheduling for all ready tasks T_r is $\mathcal{O}(T_r(T_r.P_s))$. In general, the time complexity is quadratic to the number of tasks processed. The evaluation results show that MW-DBS outperformed the success rate of FDWS [43] and its variants by 43% on workloads of 50 multiple real-world application workflows. The detailed experimental design for MW-DBS algorithm is depicted in Table. 2.6.

The latest work was the Multi-QoS Profit-Aware Scheduling (MQ-PAS) algorithm [56]. MQ-PAS was designed not only for the cloud computing environment but also for the general utility-based distributed system. Task ranking and selection complexity in MQ-PAS is $\mathcal{O}(T_w.C_s)$ for all tasks in a workflow T_w given a set of static cloud instances C_s. Meanwhile, the quality measure in cloud instances selection is $\mathcal{O}(T_r.C_s)$. Furthermore, the complexity of resource selection and task scheduling for all ready tasks T_r is $\mathcal{O}(T_r(T_r.C_s))$. In general, the time complexity is quadratic to the number of tasks processed. The evaluation results show that MQ-PAS outperformed the

Table. 2.7: Experimental design of MQ-PAS algorithm

Experiment Type	Simulation using SimGrid [53]	
Grid Settings	Number of processors	: 20–64 processors
	Platforms derivation	: Based on Grid5000 deployed in France [54]
Workload	Synthetic workflows	: Randomly generated based on Juve et al. [27]
	Number of workflows	: 50 workflows
	Arrival intervals	: 10–50% elapsed time of the last job
Workflow Properties	Node (40–120)	: Number of task on the largest level
	Regularity (0.2–0.8)	: Uniformity of the number of task in each level
	Density (0.2–0.8)	: Number of edges between two levels
	Jumps (1–4)	: Edge connection between two consecutive levels
	Deadline (1–2)	: Based on min. and max. execution time
	Budget (0–1)	: Based on min. and max. execution cost
Performance Metrics	Planning successful rate	: Ratio of successfully planed and executed workflows

success rate of FDWS [43] by only 1%, but there was a significant 20% improvement of profit on workloads of 50 multiple real-world application workflows. The detailed experimental design for MQ-PAS algorithm is depicted in Table. 2.7.

Several variations of scheduling scenarios are covered in their works. One of the specific signatures from this group is the strategy of choosing a single ready task from workflow to compete in the scheduling cycle with the other workflows. This strategy represents the term "Fairness" that becomes the primary concern in most of their works. However, with their broad scenarios that are intended to cover the general process in a multi-tenant distributed computing systems, the different requirements in clouds from utility grids (e.g., billing period schemes, dynamic and uncertain environment) are not considered in their works.

2.3.4 Online Multiple Workflows Scheduling Framework

A group from the National Chiao-Tung University focused on developing a scheduling framework for multiple workflows scheduling. Their first algorithm called the Online Workflow Management (OWM) [52] consisted of four phases–critical path workflow Scheduling (CPWS), task scheduling, multi-processor task rearrangement, and adaptive allocation (AA). The CPWS phase ranks all tasks T_w based on their relative position in their workflows before they were submitted to the scheduling queue to create a schedule plan with $\mathcal{O}(T_w . P_s)$ time complexity. Task rearrangement took place to improve resource utilization that generates $\mathcal{O}({T_r}^2)$ time complexity whenever the

Table. 2.8: Experimental design of OWM algorithm

Experiment Type	Simulation	
Grid Settings	Number of processors	: 120 processors
Workload	Synthetic workflows	: Randomly generated
	Number of workflows	: 100 workflows
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (20–100)	: Number of task
	Shape (0.5–2.0)	: Workflow's degree of parallelism
	OutDegree (1–5)	: Maximum number of immediate descendants of a task
	CCR (0.1–2.0)	: Ratio of computation and communication time
	BRange (0.1–1.0)	: Distribution range of computation cost of tasks on processors
	WDAG (100–1000)	: Average computation cost of a workflow
Performance Metrics	Makespan	: Total execution time
	Schedule length ratio	: Ratio of makespan and critical path length
	Win	: Percentage of a workflow got the shortest makespan

plan produces the scheduling gaps. Furthermore, AA schedules the highest priority task from the queue that is constructed based on the near-optimal schedule plan which time complexity is $\mathcal{O}(T_r\,(T_r.P_s\,+\,P_s)$. In general, the time complexity is quadratic to the number of tasks processed. The evaluation results show that OWM outperformed RANK_HYBD [45] by 1.15x average speedup on workloads of 100 multiple workflows. The detailed experimental design for OWM algorithm is depicted in Table. 2.8.

In their following work, they extended OWM into Mixed-Parallel Online Workflow Scheduling (MOWS) algorithm [42]. They modified the CPWS phase using the Shortest-Workflow-First (SWF) policy combined with the critical path prioritization. Then, MOWS used priority-based backfilling to fill the hole of a schedule in the task rearrangement stage. The pre-emptive task scheduling policy was introduced in the AA phase, so the algorithm allowed the system to schedule the shortest workflow first and stopped it when higher priority workflow was ready to run. The difference between MOWS and OWM is the task rearrangement phase, which uses the priority-based backfilling that takes a lower time complexity of $\mathcal{O}(T_r)$. In general, the time complexity is quadratic to the number of tasks processed. The evaluation results show that MOWS outperformed OWM [52] by 1.25x average speedup on workloads of 100 multiple workflows. The detailed experimental design for MOWS algorithm is depicted in Table. 2.9.

Both OWM and MOWS utilize periodic scheduling, which periodically creates a schedule plan for a set of ready tasks before submitting it to the scheduling queue. In this way, the algorithm can

Table. 2.9: Experimental design of MOWS algorithm

Experiment Type	Simulation	
Workload	Synthetic workflows	: Randomly generated
	Number of workflows	: 100 workflows
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (20–100)	: Number of task
	Shape (0.5–2.0)	: Workflow's degree of parallelism
	OutDegree (1–5)	: Maximum number of immediate descendants of a task
	CCR (0.1–2.0)	: Ratio of computation and communication time
	BRange (0.1–1.0)	: Distribution range of computation cost of tasks on processors
	WDAG (100–1000)	: Average computation cost of a workflow
Performance Metrics	Turnaround time	: Total execution and waiting time
	Schedule length ratio	: Ratio of makespan and critical path length
	Ratio of Shortest turnaround time	: Ratio of a workflow got the shortest turnaround time

produce better scheduling results without having intensive computation beforehand. However, this approach may still create a bottleneck if the number of ready tasks in the pool increases. Implementing a strategy to create a fairness scenario when selecting ready tasks to reduce the complexity of calculating a schedule plan may work to enhance this scheduling framework.

2.3.5 Real-time Multiple Workflows Scheduling

One of the active groups that focused on real-time and uncertainty aspects of multiple workflows scheduling was the group from The Aristotle University of Thessaloniki, Greece. They did impressive works on multiple workflows scheduling that explicitly addressed the uncertainty in cloud computing environments.

Their first work was the Earliest Deadline First with Best Fit (EDF_BF) algorithm [57]. EDF policy was used for the task selection phase, and the BF was the strategy for exploiting the schedule gap. EDF_BF incorporated schedule gap exploitation that can be identified through the estimated position of a task's execution in a specified resource. From all of the possible positions, the algorithm exploited the holes using a bin packing technique to find the best fit for a task's potential position. The result can also be used to determine which resource should be selected for that specific task. Given a set of ready tasks T_r processed each time and static processor P_s available, task selection complexity in EDF_BF is $\mathcal{O}(T_r . P_s)$. Meanwhile, the processor selection and schedule gap exploitation is $\mathcal{O}(T_r(T_r + T_r . P_s)$. In general, the complexity is quadratic to the number of tasks processed. The evaluation results show that EDF_BF outperformed the guarantee ratio (i.e., success rate) of its variants with HLF (Highest-Level First) and LSFT (Least-Space-Time First)

Table. 2.10: Experimental design of EDF_BF algorithm

Experiment Type	Simulation	
Grid Settings	Number of processors	: 32 processors
	Heterogeneity level (0–2)	: Difference in processors' speed
	Mean execution rate of processors ($\overline{\mu}=1$)	: Processors' speed
	Mean data transfer rate ($\overline{v}=1$)	: Data transfer's speed
Workload	Synthetic workflows	: Randomly generated based on Stavrinides and Karatza [58]
	Number of workflows	: 100 workflows
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (1–64)	: Number of tasks
	Relative deadline (1–2)	: Based on critical path length (CPL)
	CCR (0.1–10)	: Ratio of computation and communication time
Performance Metrics	Job guarantee ratio	: Ratio of successfully executed workflows

policies on task selection by an average of 10%. The detailed experimental design for EDF_BF algorithm is depicted in Table. 2.10.

Another work was the algorithm called the Earliest Deadline First with Best Fit and Imprecise Computation (EDF_BF_IC) [59], which extended the previous algorithm with imprecise computation. The imprecise computation was firstly introduced in [58] to tackle the problem in a real-time environment that was often needed to produce an early proximate result within a specified time limit. The imprecise computation model is implemented by dividing the task's components into a mandatory and optional component. A task is considered meeting the deadline if its mandatory part was completed, while the optional component may be fully executed, partially executed, or skipped. The evaluation results show that EDF_BF_IC outperformed the success rate of its baseline EDF by an average of 16% and cost-saving improvement by 12%. The detailed experimental design for EDF_BF_IC algorithm is depicted in Table. 2.11.

Furthermore, this group explored data-locality and in-memory processing for multiple workflow scheduling [61]. In this case, they combine the EDF_BF algorithm with a distributed in-memory storage solution called Hercules [62] to evaluate a different way of communication of workflow tasks. They consider two different communication scenarios, communication through a network, and via temporary files utilizing the Hercules in-memory storage solution. The results show that scheduling performance increased when the I/O to computation ratio was reduced by using in-memory storage, which enforces the locality of data. The evaluation results show that the application completion ratio (i.e., success rate) improves as the tardiness bound (i.e., soft deadline ratio) increases while the average makespan deteriorates. In addition, the average makespan of the

Table. 2.11: Experimental design of EDF_BF_IC algorithm

Experiment Type	Simulation	
Cloud Settings	Number of VMs	: 64 VMs
	Heterogeneity level (0.5)	: Difference in VMs' computational capacity
	Mean VM execution rate ($\overline{\mu}$ =1)	: VM's computational capacity
	Mean data transfer rate ($\overline{v}$ =1)	: Data transfer's speed
	Price per time unit ($0.01)	: VM leased fee per time unit
Workload	Synthetic workflows	: Randomly generated based on Stavrinides and Karatza [60]
	Number of workflows	: 10^5 workflows
	Arrival intervals ($\lambda = 0.2$)	: Poisson distribution
Workflow Properties	Node (1–64)	: Number of tasks
	Relative deadline (1–2)	: Based on critical path length (CPL)
	CCR (0.1–10)	: Ratio of computation and communication time
Performance Metrics	Overal provided Quality of Service	: Guarantee ratio × average result precision
	Average cost per application	: Average makespan × price per time unit

Table. 2.12: Experimental design of EDF_BF *In-Mem* algorithm

Experiment Type	Simulation	
Grid Settings	Number of processors	: 64 processors
	Heterogeneity level (0–2)	: Difference in processors' speed
	Mean execution rate of processors ($\overline{\mu}$ =1)	: Processors' speed
	Mean data transfer rate ($\overline{v}$ =1)	: Data transfer's speed
	Mean file system throughput rate ($\overline{\rho}$ =1)	: File system access' speed
Workload	Synthetic workflows	: Randomly generated based on Stavrinides and Karatza [58]
	Number of workflows	: 10^6 workflows
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (1–64)	: Number of tasks
	Relative deadline (1–2)	: Based on critical path length (CPL)
	CCR (0.1–10)	: Ratio of computation and communication time
	IOCR (0.25–1)	: Ratio of I/O and communication time
Performance Metrics	Application completion ratio	: Ratio of successfully completed workflows
	Application guarantee ratio	: Ratio of completed workflows within the deadline
	Tardiness	: Exceeding degree of the expected completion time

completed workflows improves as the I/O activities decrease. The detailed experimental design for EDF_BF *In-Mem* algorithm is depicted in Table. 2.12.

Despite the variation, their algorithms' main idea is to schedule all ready tasks using EDF policy for resources that can allow the tasks to finish at their earliest time. The algorithm maintains a local queue for each resource and then optimizes the local allocated queue using gaps filling techniques and, in one of the works, manipulates a small portion of the tasks that may have a little significance (i.e., imprecise computation). Their algorithms are designed for a multi-tenant system with a static number of resources. Therefore, the design may not be suitable for clouds–in which is suffered most by the uncertainty problems–where the auto-scaling of resources is possible.

Table. 2.13: Experimental design of OPHC-TR algorithm

Experiment Type	Real experiments	
Workload	Medical research application	: Based on the study by Watson [64]
	Number of workflows	: 1000 workflows
	Arrival intervals (200–700)	: Poisson distribution
Workflow Properties	Node (9–9000)	: Number of task
	Deadline (0–1)	: Based on workflow execution time in a dedicated fast resources
	Private instance limitation	: Private cloud resource restriction for a workflow
Performance Metrics	Cost	: Cloud resource monetary cost

2.3.6 Adaptive and Privacy-aware Multiple Workflows Scheduling

A group from The University of Sydney introduced an excellent work of multiple workflows scheduling that concerned with the privacy of users [63]. They developed two algorithms; Online Multiterminal Cut for Privacy in Hybrid Clouds using PCP Ranking (OMPHC-PCPR) and Online Scheduling for Privacy in Hybrid Clouds using Task ranking (OPHC-TR). OMPHC-PCPR was merging multiple workflows into one single workflow before scheduling. Hence, this solution is out of our scope. However, the other one, OPHC-TR, uses an approach that is inclusive of our study. Both algorithms calculate the privacy level of each workflow before they decide to schedule them in private or public clouds. The private clouds are used mainly for the workflow that comprised a high level of privacy parameters.

The main differences between the two algorithms are their input. While OMPHC-PCPR considers a single merged workflow from several workflows, OPHC-TR processes each task using a rank mechanism to decide which tasks are submitted into the scheduling queue. Given a set of tasks in a workflow T_w and static processors P_s available, task ranking and selection complexity in OPHC-TR is $\mathcal{O}(T_w . P_s)$. Meanwhile, the resource selection and task scheduling for all ready tasks T_r is $\mathcal{O}(T_r(T_r . P_s))$. In general, the time complexity is quadratic to the number of tasks processed. The evaluation results show that OMPHC-PCPR outperformed the cost-saving of the OPHC-TR algorithm by 50%. However, the overhead of merging several workflows in OMPHC-PCPR is not being evaluated thoroughly. Such an approach may result in a bottleneck when the workflows arriving, reach a certain high number. The detailed experimental design for OPHC-TR algorithm is depicted in Table. 2.13.

Another work from this group was the DGR algorithm [41]. This algorithm uses heuristics, which started the solution with the initial reservation of resources for particular scheduling tasks

Table. 2.14: Experimental design of DGR algorithm

Experiment Type	Simulation	
Grid Settings	Number of processors	: 1000 processors
	Computing capacity	: Normalized value (1–10)
	Computing speed	: Normalized value (1–8)
	Network bandwidth	: Normalized value (1–8)
Workload	Synthetic workflows	: Randomly generated
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (20–100)	: Number of task
	Shape (0.5–2.0)	: Workflow's degree of parallelism
	CCR (0.1–10)	: Ratio of computation and communication time
	Tasks' length (10–800)	: Tasks' execution time based on a time unit distribution
Performance Metrics	Makespan	: Total execution time
	Makespan speedup	: Makespan improvement percentage against HEFT
	Acceptance rate	: Percentage of successfully executed workflows
	Resource utilization	: Percentage of time when computational resources are busy

which time complexity is $\mathcal{O}(T_w.P_s)$ for all tasks in a workflow T_w given a set of static processors P_s. During the execution, uncertainty (i.e., performance and execution time variation) may profoundly affect the initial reservation and break the schedule plan. In this case, the algorithm reschedules the tasks to handle the broken reservation. DGR utilizes task rearrangement techniques and exploits a dynamic search tree to fix this reservation with $\mathcal{O}(T_r(P_s + T_r.P_s))$ time complexity. In general, the time complexity is quadratic to the number of tasks processed. The evaluation results show that DGR outperformed a traditional HEFT algorithm by 1.43x average speedup on workloads of 300 multiple workflows. The detailed experimental design for DGR algorithm is depicted in Table. 2.14.

2.3.7 Adaptive Dual-criteria Multiple Workflows Scheduling

Another adaptive approach in scheduling multiple workflows was an adaptive dual-criteria algorithm [65]. This algorithm used heuristics that utilized scheduling adjustment via task rearrangement. An essential strategy to this algorithm was the clustering of tasks and treated them as an integrated set in scheduling to minimize the critical data movement within tasks. Hence, any rearrangement or adjustment to fill the schedule holes involved the set of tasks to be moved. Given a set of tasks in a workflow T_w processed each time, task group after clustering T_g where $T_g \leq T_w$, and static processors P_s available, the task initial clustering process complexity is $\mathcal{O}({T_w}^2)$. Mean-

Table. 2.15: Experimental design of Adaptive dual-criteria algorithm

Experiment Type	Simulation	
Grid Settings	Number of processors	: 30 processors
Workload	Synthetic workflows	: Randomly generated based on Bharathi et al. [66]
	Number of workflows	: 100 workflows
	Arrival intervals	: 1–1000 seconds
Workflow Properties	Node (20–30)	: Number of task
	Workflow applications	: LIGO [7]
	CCR (0.1–10)	: Ratio of computation and communication time
Performance Metrics	Makespan	: Total execution time
	Scheduling overhead	: Delay in the scheduling process

while, the adjustment of idle time gap selection is $\mathcal{O}(T_g(T_g + P_s))$, it get a higher complexity compared to a simple Best-Fit and EFT calculation of single task due to the T_g constraint. Finally, the complexity of adaptive task group re-arrangement is $\mathcal{O}(T_g P_s)$. In general, the time complexity is quadratic to the number of tasks and task groups processed. The evaluation results show that this algorithm outperformed a similar process using traditional Best-Fit and EFT approaches up to 1.41x speedup in various scenarios. The detailed experimental design for Adaptive dual-criteria algorithm is depicted in Table. 2.15.

Since the approach used is the periodic scheduling, the frequency of scheduling cycle becomes critical. The infrequent scheduling cycle implies to the broader set of tasks to be processed, which may result in a more optimized scheduling plan but potentially required a more intensive computation for creating the plan. Meanwhile, a perpetual cycle may fasten the scheduling plan computation due to its size of tasks but may reduce the quality of a schedule. This variation is not being addressed and explored in-depth by the authors. In addition, the treatment of a cluster of tasks increases the coarse-granularity of scheduling that may widen the gaps. In this way, the task rearrangement may hardly find the holes that can be fit by a coarse-grained set of clustered tasks.

2.3.8 Multiple Workflows Scheduling on Hybrid Clouds

Another work designed for hybrid clouds was the Minimum-Load-Longest-App-First with the Indirect Transfer Choice (MLF_ID) algorithm [67]. The term Load-Longest-App had a similar concept to the critical path. Therefore, MLF_ID was a heuristic algorithm that incorporated the workflows prioritization based on their critical path and exploited the use of private clouds before

Table. 2.16: Experimental design of MLF_ID algorithm

Experiment Type	Simulation	
Cloud Settings	Number of Public Cloud VM type	: 6 types
	Number of Private Cloud Instances	: 512 CPUs
Workload	Synthetic workflows	: Randomly generated based on Bharathi et al. [66]
	Number of workflows	: 1000 workflows
Workflow Properties	Node (997–1000)	: Number of task
	Deadline (0.2–5 hours)	: Based on selected range of instance types for executing tasks
	Input dataset size	: 0–300 GB
Performance Metrics	Deadline met	: Number of workflows met their deadline
	Number of Application	: Number of workflows executed in private clouds
	Cost	: Public cloud resource monetary cost

leasing the resources in public clouds. MLF_ID partitioned the workflow based on a hierarchical iterative application partition (HIAP) to eliminate data dependencies between a set of tasks by clustering tasks with dependencies into the same set before scheduling them into either private or public clouds. The detailed experimental design for MLF_ID algorithm is depicted in Table. 2.16.

Given a set of tasks in a workflow T_w processed each time, static private cloud resources C_s, and dynamic public cloud resources C_d, the application partition complexity is $\mathcal{O}(T_w(C_s + C_d))$. Meanwhile, the ready tasks T_r scheduling which included the decision to schedule on public cloud is $\mathcal{O}(T_r.C_d)$ or private cloud is $\mathcal{O}(T_r(T_r + C_s)$. In general, the time complexity is quadratic to the number of tasks processed. The evaluation results show that the combined resources of hybrid clouds can minimize the total execution cost when the number of workflows can be allocated as much as possible to the private resources. However, the private cloud capacity is restrained as the scaling process is not as simple as such an approach in public clouds.

The use of hybrid clouds in this work is emphasized to extend the computational capacity when the available on-premises infrastructure (i.e., private clouds) are not enough to serve the workloads. Firstly, the tasks are scheduled for private clouds, and whenever the capacity is not possible to process, they are being transferred to public clouds. Even though the tasks have been partitioned to make sure that the data transfer between them is minimum, the decision to move to public clouds evokes a possible transfer overhead problem. Therefore, some improvements can be made by implementing a policy to decide whether a set of tasks is considered impractical to process in private clouds that include some intelligence, which can be designed to predict the possible overhead in the future of the system. In this way, instead of directly transferring the execution to the public clouds that incite not only the additional cost but also the transfer overhead, the algorithm can decide whether it should move the execution or delay the process waiting for the next available resources.

2.3.9 Proactive and Reactive Scheduling for Multiple Workflows

Another group that focused on real-time and uncertainty problems in scheduling was a group from The National University of Defense Technology, China. They proposed the algorithms that dynamically exploited proactive and reactive methods in scheduling.

Their first work was the Proactive Reactive Scheduling (PRS) algorithm [68]. The proactive phase calculated the estimated earliest start and the execution time of tasks and then scheduled them dynamically based on a list-based heuristic. This method has been incorporated into many algorithms for multiple workflows scheduling. However, using only the proactive method was unable to tackle the uncertainties (e.g., performance variation, overhead delays) that led to sudden changes in the system. Then, PRS introduced a reactive phase whenever two disruptive events occurred (i.e., arrival of new workflow and finishing time of a task). The reactive phase was triggered by two disruption events to update the scheduling process based on the latest system status. Given a set of tasks in a workflow T_w processed each time and dynamic cloud resources C_d available, the time complexity of task ranking is $\mathscr{O}(T_w)$. Meanwhile, the VM selection and task scheduling for all ready tasks T_r is $\mathscr{O}(T_r(T_r.C_d)$. In general, the time complexity is quadratic to the number of tasks. The evaluation results show that PRS outperformed the cost-savings of modified SHEFT [69] and RTC [70] algorithms for multiple workflows by 50.94% and 67.23% respectively on workloads of 1000 multiple workflows. The detailed experimental design for PRS algorithm is depicted in Table. 2.17.

They then extended PRS into Event-driven and Periodic Rolling Strategies (EDPRS) algorithm [71]. EDPRS tackled a flaw in PRS, that, if none of the two disruption events happened, the scheduling process could not be pushed forward. They introduced a periodic rolling strategy (i.e., scheduling cycle) that drove the re-iteration of the schedule. In this way, albeit no disruption events occurred, the algorithm repeated their scheduling activities after a specific periodic rolling time. In general, the time complexity is similar to the PRS algorithm. The evaluation results show that EDPRS outperformed the cost-savings of modified SHEFT [69] and RTC [70] algorithms for multiple workflows by 12.98% and 21.57% respectively. Both PRS and EDPRS work well in

Table. 2.17: Experimental design of PRS algorithm

Experiment Type	Simulation using CloudSim [72]	
Cloud Settings	Number of VM type Network bandwidth Cloud billing period	: 6 types : 1 Gbps : 1 hour
Workload	Synthetic workflows Number of workflows Arrival intervals	: Randomly generated based on Juve et al. [27] : 1000 workflows : Poisson distribution
Workflow Properties	Node (30–100) Deadline	: Number of task : Generated based on the fastest execution time
Performance Metrics	Cost Resource utilization Deviation	: Cloud resource monetary cost : Percentage of time when computational resources are busy : Cost of time deviation between predicted and actual finish time

Table. 2.18: Experimental design of EDPRS algorithm

Experiment Type	Simulation using CloudSim [72]	
Cloud Settings	Number of VM type VM provisioning delay Cloud billing period	: 6 types : 120 seconds : 1 hour
Workload	Synthetic workflows Number of workflows Arrival intervals	: Randomly generated based on Juve et al. [27] : 1000 workflows : Poisson distribution
Workflow Properties	Node (30–100) Deadline	: Number of task : Generated based on the fastest execution time
Performance Metrics	Cost Resource utilization Deviation	: Cloud resource monetary cost : Percentage of time when computational resources are busy : Cost of time deviation between predicted and actual finish time

handling the uncertainty in cloud computing environments. The detailed experimental design for EDPRS algorithm is depicted in Table. 2.18.

This group also worked on energy-efficient multiple workflow scheduling algorithms. Their work was the Energy-Efficient Online Scheduling (EONS) algorithm [73]. EONS was different from the other energy-efficient scheduling algorithms due to its focus on fast and real-time oriented scheduling. EONS utilized simple auto-scaling techniques to lower energy consumption instead of optimizing energy usage using techniques such as VM live migration and VM consolidation. The scaling method used simple heuristics that considered the load of the physical host and the hardware efficiency. Given a set of tasks in a workflow T_w processed each time and dynamic cloud resources C_d available, task ranking complexity in EONS is $\mathcal{O}(T_w)$. Meanwhile, the VM selection

Table. 2.19: Experimental design of EONS algorithm

Experiment Type	Simulation using CloudSim [72]	
Data center Settings	Number of Server type	: 10 types
	Number of VM types	: 10 types
	CPU resource requirements	: 200–2000 MHz
	Inter-VM bandwidth	: 1 Gbps
	VM provisioning delay	: 90 seconds
Workload	Synthetic workflows	: Randomly generated based on Juve et al. [27]
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (30–100)	: Number of task
	CCR (0.5–5.5)	: Ratio of computation and communication time
Performance Metrics	Resource utilization	: Percentage of time when computational resources are busy
	Energy Consumption	: Data center energy consumption

and task scheduling for all ready tasks T_r is $\mathcal{O}(T_r(T_r.C_d)$. In general, the time complexity is quadratic to the number of tasks processed. The evaluation results show that EONS outperformed the energy-savings of modified EASA [74] and ESFS [75] algorithms for multiple workflows by 45.64% and 35.98% respectively. The detailed experimental design for EONS algorithm is depicted in Table. 2.19.

Another work from this group addressed the failure in multiple workflows scheduling. The algorithm, called FASTER [76], utilized the primary backup technique to handle the failure. To the best of our knowledge, this is the only fault-tolerant algorithm for multiple workflows scheduling. As part of the pre-processing phase, they scheduled two copies of a task (i.e., primary and backup copies) based on the FCFS policy. The workflows were accepted for execution when both primary and backup copies successfully met their deadlines. Whenever a task was not able to meet its deadline, the algorithm re-calculates its earliest start time. This estimation takes $\mathcal{O}({T_w}^2)$ given a set of tasks in a workflow T_w processed each time. FASTER ensured that the primary copy was distributed among all available hosts as part of its fault-tolerant strategy. This heuristic requires periodic scanning of all VMs C_d within the available physical host H_d in the system. The complexity of host monitoring phase is $\mathcal{O}(H_d(C_d.T_w))$ for each primary and backup type of tasks. In general, the time complexity is quadratic to the number of tasks. The evaluation results show that FASTER outperformed the modified eFRD [77] algorithm for multiple workflows by 239.66% in terms of guarantee ratio (i.e., success rate) and 63.79% in terms of resource utilization. The detailed experimental design for FASTER algorithm is depicted in Table. 2.20.

Table. 2.20: Experimental design of FASTER algorithm

Experiment Type	Simulation using CloudSim [72]	
Cloud Settings	Number of VM type	: 4 types
	VM Provisioning delay	: 105 seconds
	Cloud billing period	: 1 hour
Workload	Synthetic workflows	: Randomly generated based on Juve et al. [27]
	Number of workflows	: 400 workflows
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (50–500)	: Number of task
	Deadline	: Uniformly distributed based on minimal execution time
Performance Metrics	Guarantee ratio	: Percentage of successfully executed workflows
	Host active time	: Total active time of all hosts
	Ratio of task time over hosts time	: Ratio of tasks' execution time over hosts active time

Their next algorithms were called ROSA [78] and CERSA [79]. These algorithms were the improvement of PRS and EDPRS algorithms that specifically tackle the uncertainties in executing multiple real-time workflows. While their previous algorithm EDPRS relies on a periodic trigger to clear a task pool beside the arrival of new workflows, ROSA and CERSA initiate the scheduling based on specific disturbance events. ROSA defined triggering events like the arrival of new workflows and the completion of a task in a particular cloud instance. On the other hand, CERSA added the arrival of the urgent task as one of the triggering events. In general, CERSA and ROSA time complexity is quadratic, similar to PRS and EDPRS. The evaluation results show that in terms of monetary cost, ROSA outperformed EPSM [80] and CWSA [17] by 10.07% and 23.18% while CERSA outperformed CWSA [17] and OPHC-TR [63] by 8.31% and 17.22% respectively. The detailed experimental design for ROSA and CERSA algorithms is depicted in Table. 2.21 and Table. 2.22 respectively.

These algorithms emphasize a specific strategy to handle real-time scenarios by using an immediate scheduling approach, which includes the update strategy to adapt to changes dynamically. However, this dynamic approach, especially on the energy-efficient and fault-tolerant problem, can be improved by optimizing the VM placement since the algorithms may have access to the information of the physical infrastructure.

2.3.10 Energy Aware Scheduling for Multiple Workflows

A group from Nanjing University, China, proposed an algorithm for multiple workflows scheduling that was called EnReal, an energy-aware resource allocation method for workflow in the cloud

Table. 2.21: Experimental design of ROSA algorithm

Experiment Type	Simulation	
Cloud Settings	Number of VM type	: 4 types
	Network bandwidth	: 1 Gbps
	VM provisioning delay	: 30 seconds
	Cloud billing period	: 1 hour
Workload	Synthetic workflows	: Randomly generated based on Juve et al. [27]
	Number of workflows	: 1000 workflows
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (25–100)	: Number of task
	Deadline	: Generated based on the fastest execution time
Performance Metrics	Cost	: Cloud resource monetary cost
	Makespan	: Total execution time

Table. 2.22: Experimental design of CERSA algorithm

Experiment Type	Simulation	
Cloud Settings	Number of VM type	: 4 types
	Cloud billing period	: 1 hour
Workload	Synthetic workflows	: Randomly generated based on Juve et al. [27]
	Number of workflows	: 1000 workflows
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (30–100)	: Number of task
	Deadline	: Generated based on the fastest execution time
Performance Metrics	Cost	: Cloud resource monetary cost
	Resource utilization	: Percentage of time when computational resources are busy

environment [81]. While the previous energy-aware algorithm–EONS–utilized auto-scaling techniques to lower the energy consumption, EnReal exploited the VM live migration-based policy. The algorithm partitioned all of the ready tasks in the queue based on their requested start time and allocated them to the resources on the same physical machine. The adjustment was made whenever a load of physical machine was exceeding the threshold, and then, VM live migration took place. The detailed experimental design for EnReal algorithm is depicted in Table. 2.23.

EnReal also adjusted the VM allocation dynamically whenever a task was finished. Combined with the physical machine resource monitoring, the global resource allocation method emphasized the platform's energy saving. However, its partitioning method did not consider the dependencies between tasks that imply a data transfer overhead when they were allocated to different physical

Table. 2.23: Experimental design of EnReal algorithm

Experiment Type	Simulation	
Data center Settings	Number of Server type	: 4 types
	Energy consumption rate	: 86–342 W
Workload	Synthetic workflows	: Randomly generated based on Liu et al. [82]
	Number of workflows	: 50–300 workflows
Workflow Properties	Node (5–25)	: Number of task
	Resource requirement (1–15)	: Number of required VMs for each task
	Task length (0.1–5.0)	: Task execution time in hours
Performance Metrics	Resource utilization	: Percentage of time when computational resources are busy
	Energy Consumption	: Data center energy consumption

machines. The energy-aware resource allocation policy in EnReal should have complemented by an ability to aware of data-locality. This policy not only minimizes energy consumption but also improves the scheduling results in terms of total execution cost and makespan. In general, the most intensive phase is resource monitoring that takes quadratic time complexity. Furthermore, the performance evaluation results show that EnReal outperformed the modified energy-aware Greedy-D [83] algorithm in terms of energy efficiency by 18% on average.

2.3.11 Monetary Cost Optimization for Workflows on Commercial Clouds

A group from the National University of Singapore proposed Dyna [44], an algorithm that focuses on the clouds' dynamicity nature. They introduced a probabilistic guarantee of any defined SLAs of workflow users as it was the closest assumption to the uncertainty environment in clouds. This approach was a novel contribution since the majority of the works assumed deterministic SLAs in their algorithms. Dyna aimed to minimize multiple workflows scheduling execution cost by utilizing VMs with spot instances pricing scheme in Amazon EC2 along with its on-demand instances. Dyna started with the initial configuration of different cloud instance types and refined the arrangement iteratively to get the better scenario that minimizes the cost while meeting the deadline. In general, the time complexity is quadratic to the number of tasks. The evaluation results show that Dyna outperformed the cost of the modified MOHEFT algorithm [84] by 74% on average. The detailed experimental design for Dyna algorithm is depicted in Table. 2.24.

Dyna presents an exploration of possible cost reduction in executing multiple workflows by utilizing spot instances in Amazon EC2. Since WaaS platforms that are assumed in their work act as a service provider, the reserved instances may further reduce the cost of running the platform.

Table. 2.24: Experimental design of Dyna algorithm

Experiment Type	Simulation using CloudSim [72]	
Cloud Settings	Number of VM type	: 4 types
	Provisioning delay (on-demand instances)	: 120 seconds
	Provisioning delay (spot instances)	: 420 seconds
	Cloud billing period	: 1 hour
Workload	Synthetic workflows	: Randomly generated based on Juve et al. [27]
	Number of workflows	: 100 workflows
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (997–1000)	: Number of task
	Workflow applications	: LIGO [7], Montage [28], and Epigenomics [27]
Performance Metrics	Deadline met	: Number of workflows met their deadline
	Cost	: Cloud resource monetary cost
	Cloud instances type	: Breakdown of cloud instances type during execution

Comparison between on-demand, spot, and reserved instances in Amazon EC2 needs to be done to deepen the plausible scenario on minimizing the cost of multiple workflows in clouds.

2.3.12 Fairness Scheduling for Multiple Workflows

Fairness Scheduling with Dynamic Priority for Multi Workflow (FSDP) [85] was an algorithm proposed by a group from Dalian University of Technology, China. FSDP emphasized the fairness aspect as it incorporated slowdown metrics into their algorithm's policy. Slowdown value was the ratio of the makespan of a workflow when it was being scheduled in dedicated service to the makespan of it being scheduled in a shared environment with the other workflows—the closest slowdown value to 1, the fairest the algorithm, scheduled the workflows in the system. FSDP also included an urgency metric, a value that represented the priority of each workflow based on its deadline. The slowdown and urgency were updated periodically when a workflow finished ensuring the refinement in the scheduling process. The detailed experimental design for FSDP algorithm is depicted in Table. 2.25.

However, the fairness scenario is not explored in-depth by the authors. FSDP is only evaluated using two different workflows on a various number of resources (i.e., processor). The issue of fairness will arise when the number of submitted workflows was high enough to represent the condition of peak hour in multi-tenant distributed computing systems. In general, the time complexity is quadratic to the number of tasks processed. The evaluation results show that FSDP slightly outperformed the overall makespan of the MMHS algorithm [86].

Table. 2.25: Experimental design of FSDP algorithm

Experiment Type	Simulation	
Workload	Synthetic workflows	: Randomly generated based on Wang et al.[87]
	Number of workflows	: 25 workflows
Workflow Properties	Width (4–12)	: Number of task on the largest level
	Regularity (0.2–0.8)	: Uniformity of the number of task in each level
	Density (0.2–0.8)	: Number of edges between two levels
	Jumps (1–4)	: Edge connection between two consecutive levels
	CCR (0.1–10)	: Ratio of computation and communication time
Performance Metrics	Makespan	: Total execution time
	Win	: Percentage of a workflow got the shortest makespan

2.3.13 Scheduling Trade-off of Dynamic Multiple Workflows

A group from Hunan University presented two algorithms. The first one was the Fairness-based Dynamic Multiple Heterogeneous Selection Value (F_DMHSV) algorithm [88]. The algorithm consisted of six steps, which were task prioritization, task selection, task allocation, task scheduling, new workflow arrival handling, and task monitoring. Task prioritization used a descending order of heterogeneous priority rank value (HPRV) [89], which included the out-degree (i.e., number of successors) of the task. This prioritization complexity is $\mathcal{O}(T_w.P_s)$ for all tasks in a workflow T_w given a set of static processors P_s. The task was selected from the ready tasks pool based on the maximum HPRV. Furthermore, the task was allocated to the processor with minimum heterogeneous selection value (HSV) [89] that optimized the task allocation criteria using the combination of upward and downward ranks which creates a complexity of $\mathcal{O}(T_r.P_s)$ for all ready tasks T_r. The task, then, was scheduled to the earliest available processor with minimum HSV. The evaluation results show that F_DMHSV outperformed RANK_HYBD [45], OWM [52], and FDWS [43] algorithms by 1.37x, 1.11x, and 1.03x average speedup respectively. The detailed experimental design for F_DMHSV algorithm is depicted in Table. 2.26.

In the same year, this group published energy-efficient algorithms which combined the Deadline-driven Processor Merging for Multiple Workflow (DPMMW) algorithm that aimed to meet the deadline, and the Global Energy Saving for Multiple Workflows (GESMW) algorithm sought to lower the energy consumption [90]. DPMMW was a clustering algorithm which allocated the clustered tasks in a minimum number of processors so that the algorithm can put idle processors

Table. 2.26: Experimental design of F_DMHSV algorithm

Experiment Type	Simulation	
Workload	Synthetic workflows	: Randomly generated
	Number of workflows	: 50 workflows
	Arrival intervals	: 0–200 time units
Workflow Properties	Node (10–50)	: Number of task
	Shape (0.5–2.0)	: Workflow's degree of parallelism
	OutDegree (1–5)	: Maximum number of immediate descendants of a task
	CCR (0.1–2.0)	: Ratio of computation and communication time
Performance Metrics	Schedule length ratio	: Ratio of makespan and critical path length
	Unfairness	: Difference between a workflow's slowdown and average slowdowns
	Deadline missed ratio	: Ratio of workflows missing the deadline

Table. 2.27: Experimental design of DPMMW & GESMW algorithm

Experiment Type	Simulation	
Data center Settings	Number of processors	: 64 processors
Workload	Synthetic workflows	: Randomly generated
	Number of workflows	: 10–50 workflows
Workflow Properties	Node (40–55)	: Number of task
	Deadline (0–2)	: Increment of the workflow lower bound execution
	Workflow applications	: Gaussian elimination [46], fast fourier transform [46], : linear algebra [92], diamond graph [92], complete binary tree [92]
Performance Metrics	Deadline missed ratio	: Ratio of workflows missing the deadline
	Energy Consumption	: Data center energy consumption

into sleep mode. Meanwhile, GESMW re-assigned and adjusted the tasks to any processor with minimum energy consumption in the global scope. The combination of DPMMW and GESMW was exploited to get lower energy consumption. This approach was different from the previous two energy-efficient algorithms that focused on virtual machine level manipulation. In general, the most intensive phase in this algorithm is the invoking of the HEFT algorithm to create a baseline scheduling plan and traverse all processors, which take quadratic time complexity. Furthermore, the performance evaluation results show that DPMMW & GESMW outperformed the energy saving of the reusable DEWTS, a modified version of the DEWTS algorithm [91] by 8.1% on average. The detailed experimental design for DPMMW & GESMW algorithm is depicted in Table. 2.27.

This group presents two opposite approaches to scheduling with different objectives. In both methods, the algorithms emphasize a strategy of resource selection. In their first work, the algorithm focuses on selecting various resources to minimize the makespan. At the same time, it

Table. 2.28: Experimental design of CWSA algorithm

Experiment Type	Simulation using CloudSim [72]	
Cloud Settings	Number of VM type	: 4 types
	Provisioning delay	: 97 seconds
	Cloud billing period	: 1 hour
Workload	Synthetic workflows	: Randomly generated based on Juve et al. [27]
	Number of workflows	: 1–20 workflows
Workflow Properties	Node (30–1000)	: Number of task
	Workflow applications	: Cybershake [29] and SIPHT [93]
	Deadline (2–4)	: Based on critical path length (CPL)
Performance Metrics	Makespan	: Total execution time
	Tardiness	: Exceeding degree of the expected completion time
	Laxity	: Degree of a task's urgency execution
	Mean scheduling execution time	: Average time taken by the scheduler to execute workflow
	Resource utilization rate	: Percentage of time when computational resources are busy
	Makespan standard deviation	: Standard deviation of the workflow's makespan
	Skewness of makespan	: Symmetry measurement of the makespan distribution
	Cost	: Cloud resource monetary cost

chooses different machines with various energy efficiency to reduce energy consumption. These strategies can improve the overall result by combining them with efficient task scheduling.

2.3.14 Workflow Scheduling in Multi-tenant Clouds

Another algorithm for multiple workflows scheduling was Cloud-based Workflow Scheduling (CWSA) [17]. This work used the term "multi-tenant clouds" in its paper for describing the multi-tenancy aspect that was generally considered in cloud computing environments, whereas the definition itself was similar to the multiple workflows we used in this survey. The algorithm was intended for compute-intensive workflows applications. Hence, CWSA ignored data-related overhead and focused on compute resource management. The algorithm was aimed to minimize the total makespan, which in the result, decreased the cost of execution. In general, the time complexity of CWSA is $\mathcal{O}(T_w.C_d)$, given a set of workflow tasks T_w and a number of dynamic cloud instances C_d. The performance evaluation results show that CWSA outperformed both the makespan and cost of standard FCFS, EASY Backfilling, and Minimum Completion Time policy. The detailed experimental design for CWSA algorithm is depicted in Table. 2.28.

However, CWSA does not further optimize its cost minimization strategy using a cost-aware resource provisioning technique. CWSA auto-scales the resources using a resource utilization threshold, in which it acquires and releases the resources if their utilization exceeded or below a

Table. 2.29: Experimental design of EPSM algorithm

Experiment Type	Simulation using CloudSim [72]	
Cloud Settings	Number of VM type	: 4 types
	VM Provisioning delay	: 0–250 seconds
	Container Provisioning delay	: 0–100 seconds
	Cloud billing period	: 1 hour
Workload	Synthetic workflows	: Randomly generated based on Juve et al. [27]
	Number of workflows	: 1000–4000 workflows
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (30–1000)	: Number of task
	Deadline	: Randomly generated based on simulated execution time
Performance Metrics	Deadline met	: Number of workflows met their deadline
	Cost	: Cloud resource monetary cost
	Avg. VM utilization	: Percentage of time when a VM is busy
	Makespan/Deadline Ratio	: Ratio of actual makespan and assigned deadline
	Cloud instances type	: Breakdown of cloud instances type during execution

specific number. For example, they implemented the following rule: if the usage is $\geq 70\%$ for 10 minutes, then it is scaled-up by adding 1 VM of small size. In this case, the algorithms with cost-aware auto-scaling strategy–that specifically acquires and releases particular VMs based on the workload–may outperform CWSA that only considers overall system utilization based auto-scaling. This type of auto-scaling is not accurately provisioning resources that are tailored to the need of workloads.

2.3.15 Multi-tenant WaaS Platform

Another solution for multiple workflow scheduling was Elastic Resource Provisioning and Scheduling Algorithm for Multiple Workflows designed for WaaS Platforms (EPSM) [80]. This work used a specific term of "multi-tenant" to describe the platform for executing multiple workflows in the clouds. However, the "multi-tenant" term and "multiple workflows" can be used interchangeably in this case. The EPSM introduced a scheduling algorithm for WaaS platforms that utilized a container to bundle workflow's application before deploying it into VMs. In this way, the users can share the same VMs without having any problem related to software dependencies and libraries. The detailed experimental design for EPSM algorithm is depicted in Table. 2.29.

The algorithm consisted of two-phase, resource provisioning which included a flexible approach of scaling up and down the resources to cope with the dynamic workload of workflows, and scheduling which exploited a delay policy based on the task's deadline to re-use the cheapest resources as much as possible to minimize the cost. In the resource provisioning phase, EPSM incorporated an overhead detection in the form of provisioning delay and de-provisioning delay of the VMs. This strategy was able to reduce unnecessary costs due to violating a coarse-grain billing period of clouds. The algorithm made an update of the unscheduled tasks' deadline whenever a task finished the execution. In this way, the algorithm dynamically adapted the gap between the estimated and actual execution plans to ensure scheduling objectives. In the scheduling phase, EPSM considered re-using available VMs before provisioning the new one to minimize the delay of acquiring new VMs and possible cost minimization by re-using the cheapest VMs available. In general, the time complexity of the EPSM algorithm is quadratic to the number of tasks processed. Furthermore, the performance evaluation results show that this algorithm outperformed the cost-saving of the Dyna algorithm [44] by 19% on average for various scenarios.

2.3.16 Concurrent Multiple Workflows Scheduling

The latest work on deadline- and budget-constrained multiple workflows scheduling was Multi-workflow Heterogeneous budget-deadline-constrained Scheduling (MW-HBDCS) algorithm [94] that was introduced by a group from Guangzhou University, China. This work used the term "concurrent multiple workflows" as it emphasized the concurrent condition of multiple workflows, which means tackling several workflows that arrived at the same time or overlapped on a dense condition. MW-HBDCS was designed to improve the flaw on the previous similar algorithm, MW-DBS [55]. Significant enhancement was the inclusion of a budget in the ranking process to prioritize the tasks for scheduling. The algorithm was also designed to tackle uncertainties in the environments. In this work, the authors use the terms "consistent" and "inconsistent" environments to describe various dynamicity in multi-tenant distributed computing systems. In general, the time complexity is quadratic, similar to MWDBS time complexity. The evaluation results show that MW-HBDCS outperformed the success rate of MW-DBS by 46% and 52% on synthetic and real-world workflow applications, respectively. The detailed experimental design for MW-HBDCS algorithm is depicted in Table. 2.30.

MW-HBDCS tackles many flaws that are not considered in the previous deadline- and budget-constrained scheduling algorithms. These enhancements are the model that incorporated high uncertainties and dynamicity, the improvement of task's ranking mechanism that enclosed the budget as one of the primary constraints besides the deadline, while previously only acted as a comple-

Table. 2.30: Experimental design of MW-HBDCS algorithm

Experiment Type	Simulation using SimGrid [53]	
Grid Settings	Number of processors	: 20–64 processors
	Platforms derivation	: Based on Grid5000 deployed in France [54]
Workload	Synthetic workflows	: Randomly generated based on Juve et al. [27]
	Number of workflows	: 50 workflows
	Arrival intervals	: 10–50% elapsed time of the last job
Workflow Properties	Node (10–100)	: Number of task on the largest level
	Regularity (0.2–0.6)	: Uniformity of the number of task in each level
	Density (0.2–0.6)	: Number of edges between two levels
	Jumps (1–4)	: Edge connection between two consecutive levels
	CCR (0.1–5)	: Ratio of computation and communication time
	BRange (0.1–2)	: Distribution range of computation cost of tasks on processors
	Deadline (0.5–0.9)	: Based on min. and max. execution time
	Budget (0.4–0.8)	: Based on min. and max. execution cost
Performance Metrics	Planning successful rate	: Ratio of successfully planed and executed workflows

mentary constraint. Since the authors highly considered the budget as crucial as the deadline, it is essential to include the trade-off analysis between the values of budget and deadline related to the success rate of workflows execution. One of the techniques to such an approach is the Pareto analysis that is used for multi-objective workflow scheduling (e.g., MOHEFT [84]). However, this algorithm adopts static resource provisioning. It may not achieve optimal performance in cloud computing environments where the auto-scaling of resources is possible.

2.3.17 Scheduling Multiple Workflows under Uncertain Execution Time

The latest deadline-aware multiple workflows scheduling algorithm is NOSF [95]. This algorithm adopted a similar strategy to several previous algorithms (e.g., EDPRS, EPSM, ROSA) designed to tackle the uncertainties in cloud computing environments. The NOSF aims to minimize the cost of leasing cloud instances by optimizing resource utilization using the sharing strategy of the VM billing period. To further distribute a fair share of sub-deadlines between tasks, NOSF made use of PCP to create end-to-end scheduling of several tasks during the deadline distribution process. Therefore, traversing workflows for detecting the PCP is the most intensive phase that takes quadratic time complexity. The detailed experimental design for NOSF algorithm is depicted in Table. 2.31.

NOSF relies on the strategy to maintain the minimum growth of the leased cloud instances instead of auto-scaling the resources dynamically by eliminating future idle VMs. This strategy

Table. 2.31: Experimental design of NOSF algorithm

Experiment Type	Simulation	
Cloud Settings	Number of VM type	: 7 types
	Network bandwidth	: 100 Mbps
	VM provisioning delay	: 97 seconds
	Cloud billing period	: 1 hour
Workload	Synthetic workflows	: Randomly generated based on Juve et al. [27]
	Number of workflows	: 1000 workflows
	Arrival intervals	: Poisson distribution
Workflow Properties	Node (30–1000)	: Number of task
	Deadline	: Generated based on the fastest execution time
Performance Metrics	Resource utilization	: Percentage of time when computational resources are busy
	Deadline violation	: Percentage of exceeding workflow deadlines

may cause a problem of waiting overhead in a very dense workflows' arrival (i.e., high concurrent workflows). Combining dynamic auto-scaling and maintaining a low growth of VM leased may become an essential strategy for WaaS platforms with quite high uncertainties situation. Furthermore, the performance evaluation results show that NOSF outperformed ROSA [78] in terms of reducing the cost and deadline violation probability by an average of 50.5% and 55.7% respectively while improving resource utilization by 32.6% on average.

2.3.18 Algorithm Classification

This section presents the mapping between the state-of-the-art algorithms and their specific characteristics. Each algorithm is classified based on the taxonomy presented in Section 2.2. Furthermore, Table 2.32 displays the workload and deployment models taxonomy along with the unique keyword for each algorithm. Table 2.33 depicts the classification from the scheduling perspective, which includes the priority assignment, task scheduling, and resource provisioning models taxonomy.

2.4 Summary

This chapter presents a study on algorithms for multiple workflows scheduling in multi-tenant distributed systems. In particular, the research focuses on the heterogeneity of workloads, the model for deploying multiple workflows, the priority assignment for multiple users, the scheduling techniques for multiple workflows, and the resource provisioning strategies in multi-tenant distributed systems. It presents a taxonomy covering the focus of the study based on a comprehensive review

Table. 2.32: Taxonomy of workload and deployment model

Algorithms	Refs	Keywords	Workload Model				Deployment Model		
			Workflow Type		QoS Requirements		Non-virtualized	Virtualized	
			Homo	Hetero	Homo	Hetero		VM-based	Container-based
RANK_HYBD	[45]	Dynamic-guided scheduling	-	✓	✓	-	✓	-	-
OWM	[52]	Scheduling framework	-	✓	-	✓	✓	-	-
MOWS	[42]		-	✓	-	✓	✓	-	-
P-HEFT	[50]	Dynamic-parallel scheduling	-	✓	✓	-	✓	-	-
MQMW	[48]	Multi-QoS scheduling	-	✓	✓	-	-	✓	-
MQSS	[49]		-	✓	✓	-	✓	-	-
EDF_BF	[57]	Exploiting schedule gaps	-	✓	✓	-	✓	-	-
EDF_BF_IC	[59]		-	✓	✓	-	✓	-	-
CWSA	[17]		-	✓	✓	-	-	✓	-
FSDP	[85]	Fairness & priority	-	✓	✓	-	✓	-	-
F_DMHSV	[88]		-	✓	✓	-	✓	-	-
FDWS	[43]		-	✓	✓	-	✓	-	-
Adaptive dual-criteria	[65]	Partition-based scheduling	-	✓	✓	-	✓	-	-
MLF_ID	[67]		-	✓	✓	-	✓	-	-
OPHC-TR	[63]	Privacy constraint	✓	-	✓	-	-	✓	-
Dyna	[44]	Deadline constraint	-	✓	✓	-	-	✓	-
EPSM	[80]		-	✓	✓	-	-	-	✓
DGR	[41]	Task rearrangement	-	✓	✓	-	✓	-	-
FASTER	[76]	Fault-tolerant	-	✓	✓	-	-	✓	-
EnReal	[81]	Energy-efficient	-	✓	✓	-	-	✓	-
EONS	[73]		-	✓	✓	-	-	✓	-
DPMMW & GESMW	[90]		-	✓	✓	-	-	✓	-
PRS	[68]	Uncertainty-aware	-	✓	✓	-	-	✓	-
EDPRS	[71]		-	✓	✓	-	-	✓	-
ROSA	[78]		-	✓	✓	-	-	✓	-
NOSF	[95]		-	✓	✓	-	-	✓	-
EDF_BF *In-Mem*	[61]	Data-locality perspective	-	✓	✓	-	✓	-	-
MW-HBDCS	[94]	Deadline-budget constraints	-	✓	✓	-	✓	-	-
MW-DBS	[55]		-	✓	✓	-	✓	-	-
MQ-PAS	[56]	Profit-aware	-	✓	✓	-	-	✓	-
CERSA	[79]	Real-time scheduling	-	✓	✓	-	-	✓	-

of multiple workflows scheduling algorithms. The taxonomy is accompanied by a classification from surveyed algorithms to show the existing solution's coverage in various aspects. The current algorithms within the scope of the study are reviewed and classified to open up the problems in this area. Some descriptions and discussions of various solutions are covered in this chapter to give a more detailed and comprehensive understanding of the state-of-the-art techniques and even to get an insight into further research and development in this area.

Table. 2.33: Taxonomy of priority assignment, task scheduling, and resource provisioning model

Algorithms	Refs	Priority Assignment Model				Task Scheduling Model			Resource Provisioning Model		
		App. Type	QoS Const.	User-defined	Wf Structure	Immediate	Periodic		Static	Dynamic	
							Gap Search	Conf. Search		Workload	Performance
RANK_HYBD	[45]	-	-	-	✓	✓	-	-	✓	-	-
OWM	[52]	-	-	-	✓	-	✓	-	✓	-	-
MOWS	[42]	-	-	-	✓	-	✓	-	✓	-	-
P-HEFT	[50]	-	-	-	✓	✓	-	-	✓	-	-
MQMW	[48]	-	✓	-	-	✓	-	-	✓	-	-
MQSS	[49]	-	✓	-	-	✓	-	-	✓	-	-
EDF_BF	[57]	-	✓	-	-	-	✓	-	✓	-	-
EDF_BF_IC	[59]	-	✓	-	-	-	✓	-	✓	-	-
CWSA	[17]	-	✓	-	-	-	✓	-	-	-	✓
FSDP	[85]	-	✓	-	-	✓	-	-	✓	-	-
F_DMHSV	[88]	-	-	-	✓	-	✓	-	✓	-	-
FDWS	[43]	-	-	-	✓	✓	-	-	✓	-	-
Adaptive dual-criteria	[65]	-	✓	-	✓	-	✓	-	✓	-	-
MLF_ID	[67]	-	✓	-	-	✓	-	-	-	✓	-
OPHC-TR	[63]	✓	✓	-	-	✓	-	-	-	✓	-
Dyna	[44]	-	✓	-	-	-	-	✓	-	✓	-
EPSM	[80]	-	✓	-	-	✓	-	-	-	✓	-
DGR	[41]	-	✓	-	-	-	✓	-	✓	-	-
FASTER	[76]	-	✓	-	-	-	✓	-	-	✓	-
EnReal	[81]	-	✓	-	-	-	-	✓	-	✓	-
EONS	[73]	-	-	-	-	✓	-	-	-	✓	-
DPMMW & GESMW	[90]	-	✓	-	-	-	✓	-	-	✓	-
PRS	[68]	-	✓	-	-	✓	-	-	-	✓	-
EDPRS	[71]	-	✓	-	-	✓	-	-	-	✓	-
ROSA	[78]	-	✓	-	-	✓	-	-	-	✓	-
NOSF	[95]	-	✓	-	✓	✓	-	-	-	✓	-
EDF_BF *In-Mem*	[61]	-	✓	-	-	-	✓	-	✓	-	-
MW-HBDCS	[94]	-	✓	-	✓	✓	-	-	✓	-	-
MW-DBS	[55]	-	✓	-	✓	✓	-	-	✓	-	-
MQ-PAS	[56]	-	✓	-	✓	✓	-	-	✓	-	-
CERSA	[79]	-	✓	✓	-	✓	-	-	-	✓	-

Chapter 3

A Task-based Budget Distribution Strategy for Scheduling Workflows

This chapter proposes a budget-distribution algorithm that assigns a portion of the overall workflow budget to the individual tasks. This sub-budget then guides the dynamic scheduling process and is continuously refined to reflect any unexpected costs. The algorithm is evaluated using the simulation toolkit for clouds, CloudSim. The performance evaluation results demonstrate that in 88% of the cases, this approach achieves equal or better performance in terms of meeting the budget constraint and achieves lower execution times in 84% of the cases compared to the state-of-the-art algorithm.

3.1 Introduction

IaaS clouds offer a convenient way for WMS to access computational resources. These VMs can be accessed on-demand and users are charged only for what they use, usually in increments of a billing period defined by the provider. This flexibility and ability to easily scale the number of resources leads to a trade-off between two conflicting QoS requirements: time and cost. There has been extensive research [26] on this topic, with most works proposing algorithms that aim to minimize the total execution cost, while finishing the workflow execution before a user-defined deadline. In this chapter, we focus on optimizing the usage of resources so that the total execution time of the workflow (i.e., makespan) is minimized while meeting a budget constraint.

Various strategies can be used to achieve these scheduling objectives when deploying workflows in IaaS clouds. A popular one is using meta-heuristics to produce a static mapping of tasks to resources in advance. In this way, an estimate of the total cost and makespan of the workflow

This chapter is derived from: **Muhammad H. Hilman**, Maria A. Rodriguez, and Rajkumar Buyya. ‘Task-based Budget Distribution Strategies for Scientific Workflows with Coarse-grained Billing Periods in IaaS Clouds.‘ *In Proceedings of the 13th IEEE International Conference on e-Science (e-Science)*, Pages 128-137, 2017.

execution is known as well as the required resources and their leasing period. This technique, however, is computationally intensive and does not scale well with the number of tasks in the workflow. Also, because the schedule is produced before runtime and remains unchanged throughout the execution, these algorithms are unable to adapt to the inherent dynamicity and uncertainty of the IaaS clouds environment. Other algorithms use lighter-weight heuristics to produce static schedules to address the scalability issue. However, they still fail to respond to environmental changes. Some strategies choose to dynamically schedule the tasks as they become ready for execution to overcome the responsiveness issue. This dynamic approach enables the algorithm to scale easily and to adapt and make decisions based on the state of the system. Since the scheduling is task-based, the overall workflow budget must be distributed to each task. This budget allocation guides the scheduling process, since it determines the type of resources that can be allocated to each task as well as the time when they should be deployed. Budget distribution is a challenging problem mainly due to the pricing model offered by IaaS cloud providers.

It is not uncommon for the average execution time of tasks to be considerably smaller than the billing periods (e.g., one hour) offered by IaaS vendors. Thus, scheduling algorithms aim to efficiently utilize idle time slots on leased VMs as a cost-controlling mechanism. Coarse-grained billing periods make it difficult to estimate the portion of the budget that should be allocated to each task; the reason being that the cost of a single task must be overestimated by either rounding up its execution time to one (or more) billing periods or (potentially) underestimated by determining their cost in time units. Deciding how to factor VM provisioning delays when estimating the costs of tasks is another challenge. Some algorithms choose to consolidate tasks per workflow level and assign a corporate budget to them to be spent greedily to avoid time slots wastage. Depending on how the budget is split, this may result in the insufficient budget allocated to some levels, violating budget constraints due to tasks in a level taking longer to execute, and inefficient use of the budget. In this chapter, we explore distributing the budget to each task by rounding their cost to billing periods. We argue that this enables the algorithm to spend the budget more efficiently as it has a better awareness of the remaining budget and hence can better utilize it. Furthermore, such an algorithm can respond faster to unexpected delays. Also, to avoid underutilizing resources, this strategy is combined with policies that encourage the reuse of time slots in already-leased VMs.

Thus, we focus on distributing a portion of the budget to individual tasks and spending it only when necessary, that is when free idle time slots on existing VMs cannot be reused. Our approach considered the inherent features of clouds, such as the abundance of heterogeneous computing resources, VM provisioning delays, and the dynamic and uncertain behaviour of VMs performance. Our solution consists of two components, a budget distribution strategy and scheduling. For the budget distribution, we propose an algorithm with two variants, Fastest-First Task-based Distribution (FFTD) and Slowest-First Task-based Distribution (SFTD). For the scheduling, we adapt EPSM [80], an existing approach designed to schedule multiple workflows with deadline constraints. We modify it so that it considers a single workflow and aims to complete its execution as fast as possible with the given budget. Our simulation results demonstrate that our algorithm adapts to unexpected delays and meets the budget constraint while achieving lower makespans compared to the state-of-the-art budget distribution algorithm.

The rest of this chapter is organized as follows. Section 3.2 reviews works that are related to our discussion. Section 3.3 describes the considered resource and application models. The proposed algorithms are explained in Section 3.4 followed by their performance evaluation and a discussion of the results in Section 3.5. Finally, Section 3.6 summarizes the findings.

3.2 Related Work

The scheduling of scientific workflows in IaaS clouds has been extensively researched. The majority of existing algorithms have the objective to meet a deadline constraint and minimize the cost of renting the cloud computing infrastructures. Examples include the solutions by Mao and Humphrey [96], Abrishami et al. [97], Malawski et al. [98], Arabnejad et al. [99], Cai et al. [100], and Chen et al. [71].

Only a few of the existing algorithms focused on meeting budget constraints while minimizing the makespan. An example is the Partial Critical Paths Budget Balanced (PCP-B^2) [101] algorithm. It partitioned a workflow into pipelines of partial critical paths and found the optimal resource type that maximizes the budget utilization. Contrary to our work, PCP-B^2 assumed a time unit pricing model as opposed to the conventional model of billing periods. The Critical-Greedy [102] algorithm found a workflow's schedule by iteratively refining an initial schedule plan that encourages the use of more powerful VM types if there is budget remaining. Other works with the

Table. 3.1: Summary of related work

Strategies	**[101]**	**[102]**	**[103]**	**[104]**	**[105]**	**[106]**	**[107]**	**Ours**
Static Heuristic	✓	✓	-	-	✓	-	-	-
Static Metaheuristic	-	-	✓	✓	-	-	-	-
Dynamic Level-based	-	-	-	-	-	✓	✓	-
Dynamic Task-based	-	-	-	-	-	-	-	✓

same objectives used Particle Swarm Optimization [103] and Genetic Algorithms [104] to generate a static plan before runtime. These algorithms relied on calculating a near-optimal schedule by using computationally intensive meta-heuristic techniques. The strategy differs from our solution in that we use a lightweight, adaptive, heuristic-based dynamic approach that makes scheduling decisions at runtime based on the systems' state. The DBD-CTO [105] algorithm considered budget as a constraint. However, contrary to our solution, the deadline is also a part of its constraint.

BAGS [106] is another example of existing budget-constrained algorithm. It partitioned the workflow into bags of tasks (BoTs) that are on the same workflow level. BAGS was based on an online budget distribution strategy that guides the resource provisioning and scheduling plans of BoTs dynamically, as tasks become ready for execution. However, contrary to ours, BAGS assumed one minute billing periods that are not much longer than the average execution time of tasks. Finally, the Budget Distribution Trickling (BDT) [107] algorithm used a similar strategy by consolidating tasks on the same workflow level. The budget is distributed to each level, and the algorithm trickles down any remaining budget to the next level. It assumes an hourly billing period, but ignores the VMs performance variation. BDT explored several budget distribution strategies according to parameters such as the number of tasks in the level and the number of levels in the workflow. The algorithms differ from ours in that our solution scheduled tasks independently when they are ready for execution, that is, whenever the task's parents have finished executing, and the input data is available. We present a summary of the works in Table 3.1.

3.3 Application and Resource Model

VMs are leased using an on-demand pricing model and are charged per billing period bp, with any partial usage being rounded up to the nearest billing period. Our work considered a heterogeneous environment with various VM types vmt that have different processing capacity PC_{vmt} and different cost per billing period c_{vmt}. The processing capacity of a VM is measured in Million of Instruction per Second (MIPS). We assumed the CPU performance of VMs is not stable as reported by Leitner and Cito [4] and that providers advertise the maximum CPU capacity achievable by VMs. Furthermore, we assumed an unlimited VMs could be leased from the provider.

The runtime of a task t in a VM of type vmt is denoted as RT_{vmt}^{t} and is calculated based on the task's size S_t and the processing capacity PC_{vmt} of the VM. This definition is shown in Eq. 3.1.

$$RT_{vmt}^{t} = S_t / PC_{vmt} \tag{3.1}$$

Note that this value is an estimate, and our approach did not rely on it being completely accurate. We assumed the S_t, is always available to the scheduler and is measured in Millions of Instructions (MI). Also, we assumed that VMs with more CPU capacity are more expensive to lease than VMs with less capacity. In this way, the task runtime estimated using the cheapest VM type leads to the slowest runtime but potentially the lowest cost.

We considered a global storage system such as Amazon S3 for data sharing between tasks. Each task retrieves its input data D_{in}^{t} from the global repository and stores its output data D_{out}^{t} on the same. The read and writing speeds of the global storage are stated as GS_{read} and GS_{write} respectively. Additionally, each VM has a bandwidth B_{vmt} associated with it. This bandwidth and the I/O speeds of the storage system change over time, based on the number of transactions Tr running at time t. This is depicted in Eqs. 3.2, 3.3 and 3.4.

$$B_{vmt}(t) = (B_{vmt} / Tr_t) \tag{3.2}$$

$$GS_{read}(t) = (GS_{read} / Tr_t^{read}) \tag{3.3}$$

$$GS_{write}(t) = (GS_{write} / Tr_t^{write}) \tag{3.4}$$

The time it takes to retrieve the input data from the global storage to a VM is shown in Eq. 3.5.

$$T_{vmt}^{D_{in}^{t}} = (D_{in}^{t} / B_{vmt}) + (D_{in}^{t} / GS_{read}) \tag{3.5}$$

Similarly, the time it takes to transfer the output data from a VM into the global storage is shown in Eq. 3.6.

$$T_{vmt}^{D_{out}^{t}} = (D_{out}^{t}/B_{vmt}) + (D_{out}^{t}/GS_{write}) \tag{3.6}$$

We considered a model in which the global storage system and VMs are located in the same region or availability zone. Hence, data transfer between storage and VMs is free of charge, as is the case for most IaaS providers. Nevertheless, we assumed that the output data of a task is also stored on the VM's local storage. In this way, child tasks executing on the same VM did not need to read their input data from the global storage. By implementing this mode, the amount of time spent transferring data can be considerably reduced. Hence, the total processing time PT_{vmt}^{t} of a task on a VM of type *vmt* is shown in Eq. 3.7.

$$PT_{vmt}^{t} = RT_{vmt}^{t} + T_{vmt}^{D_{in}^{t}} + T_{vmt}^{D_{out}^{t}} \tag{3.7}$$

Furthermore, the cost C_{vmt}^{t} of a task that runs on a VM of type *vmt* considering the VMs provisioning delay T_{pdelay} and deprovisioning delay T_{ddelay} is shown in Eq. 3.8.

$$C_{vmt}^{t} = \lceil (PT_{vmt}^{t} + T_{pdelay} + T_{ddelay})/bp \rceil * c_{vmt} \tag{3.8}$$

3.4 Scheduling Algorithm

The algorithm consists of two steps, budget distribution, then it is followed by resource provisioning and scheduling. First, the workflow's budget is distributed to each task using FFTD and SFTD. Furthermore, the tasks are selected based on the ascending order of their Earliest Finish Time (EFT), and the VMs are provisioned based on the task's sub-budget whenever there are no idle VMs to reuse.

Our resource provisioning and scheduling strategy are based on the EPSM algorithm [80], a dynamic heuristic-based algorithm for Workflow-as-a-Service (WaaS) frameworks that can handle a continuously arriving workload of heterogeneous workflows. The algorithm's objective is meeting the deadline constraint of each workflow while minimizing the cost. To achieve this, EPSM uses containers as a means to reuse VMs across different workflows. We modified it to consider budget as a constraint and schedule a single workflow, eliminating the need for containers.

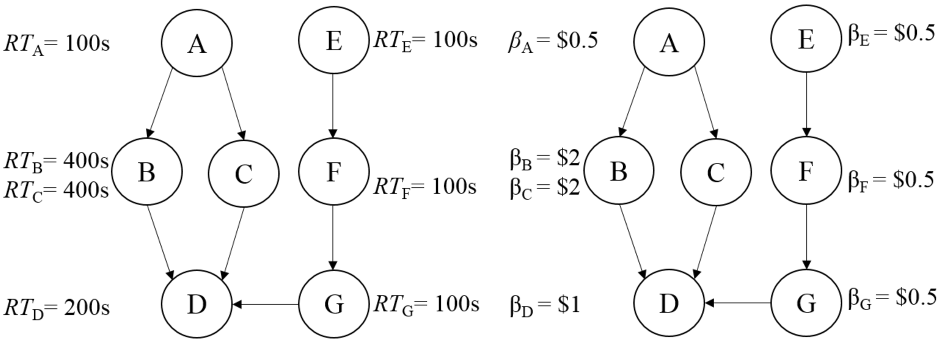


Fig. 3.1: Sample of budget insufficiency scenario

3.4.1 Budget Distribution

The amount of available budget drives the scheduling process. It determines the resources that can be used to run a task and hence has a direct impact on the workflow's makespan. Our strategy is based on the intuitive idea that by choosing the fastest VMs that are affordable within the budget, the probability that a task's runtime exceeds the billing period of a VM because of performance degradation is decreased. In this way, the likelihood of having higher costs and exceeding the budget is also decreased, since the chances of incurring additional billing periods are reduced.

This budget distribution is a challenging problem, mainly due to the coarse-grained billing periods enforced by IaaS vendors. For instances, in some cases, assigning a budget to a task, based solely on its runtime estimation without considering billing periods, may lead to budget insufficiency. This is a condition in which the task's sub-budget is not enough to provision a new VM for it, since the cost of the VM is higher than the estimated value.

Consider the illustration in Fig. 3.1. Suppose there are two VM types available, a small type which costs \$1/hour and a large type that costs \$3/hour. Then, we have a workflow that consists of seven tasks with the estimated runtime for each task being $RT_A = 100s$, $RT_B = 400s$, $RT_C = 400s$, $RT_D = 200s$, $RT_E = 100s$, $RT_F = 100s$, and $RT_G = 100s$. The budget for this workflow is \$7. If we distribute the budget based on the task's runtime and ignore the billing period, the budget for tasks A, E, F and G will be insufficient because the allocated budget (\$0.5) would be less than the cost of small type VM (\$1). Since task A and E are the entry tasks of workflow, if their sub-budgets are insufficient to provision the resources, the workflow cannot be further executed.

Our budget distribution algorithm is based on the execution order of tasks. Entry tasks in the first level of the workflow are executed first, followed by their children on the next level, and so on. Hence, we assign each task a level based on the Deadline Top Level (DTL) [108] technique, as seen in Eq. 3.9, as opposed to Deadline Bottom level (DBL) [109] which starts the level allocation from the exit task.

$$level(t) = \begin{cases} 0 & ifPred(t) = \emptyset \\ \\ \max\limits_{p \in Pred(t)} level(p) + 1 & otherwise \end{cases} \tag{3.9}$$

Let us consider the example of Fig. 3.1, DTL allocates task A and task E to the same level (1) while DBL assigns the tasks to a level starting from task D as level (1). Consequently, using DBL, task A is assigned to level (3), which would be different from the level of task E (4). Although a workflow is being processed starting from the entry tasks, allocating them into different levels has an impact on those algorithms that execute the tasks based on their level. Also, to determine the tasks' order at a level, we sort them based on the ascending order of their Earliest Finish Time (EFT), as shown in Eq. 3.10.

$$eft(t) = \begin{cases} PT^t_{vmt} & ifPred(t) = \emptyset \\ \\ \max\limits_{p \in Pred(t)} eft(p) + PT^t_{vmt} & otherwise \end{cases} \tag{3.10}$$

The tasks in the workflow in Fig. 3.1 were sorted in the following order: A [level(1)] → E [level(1)] → F [level(2)] → B [level(2)] → C [level(2)] → G [level(3)] → D [level(4)]. The budget distribution algorithm then iterates over this sorted list. It distributes the budget to each task, while considering the task's estimated runtime and the cost per billing period of each VM type. As a result, the sub-budget allocated to a task is equivalent to at least the cost of one full billing period. With this approach, we are overestimating the cost of a task, and as a result, it is possible that several tasks do not get any sub-budget allocation. In these cases, the algorithm delays the tasks with no budget so that they can reuse existing idle VMs. The budget distribution algorithm is shown in Algorithm 1.

The algorithm used two approaches to estimate the task sub-budget based on the VM type chosen, Fastest First Task-based Budget Distribution (FFTD) and Slowest First Task-based Budget

Algorithm 1 Budget Distribution

```
procedure DISTRIBUTEBUDGET($\beta$, $T$)
    $S$ = task's estimated execution order
    for each task $t \in T$ do
        $allocateLevel(t, l)$
        $initiateBudget(0, t)$
    for each level $l$ do
        $T_l$ = set of all tasks in level $l$
        sort $T_l$ based on ascending Earliest Finish Time (EFT)
        $put(T_l, S)$
    while $\beta > 0$ do
        $t = S.poll$
        $vmt$ = chosen VM type
        $allocateBudget(C^t_{vmt}, t)$
        $\beta = \beta - C^t_{vmt}$
```

Distribution (SFTD). The FFTD approach selects the fastest VM type that is affordable within the workflow's budget. Since the algorithm allocates the fastest resources to the earlier tasks, their successors have the opportunity to reuse these VMs, which may be faster than what they can afford. Hence, it also increases the possibility of obtaining lower task processing times, even though it may involve a waiting delay for the VMs to become available. On the contrary, the SFTD approach chooses the cheapest resources for the tasks. Whenever there is any remaining additional budget after all tasks are allocated sub-budgets, the algorithm uses this extra budget to greedily lease VMs with more CPU capacity than what is affordable by the individual task's sub-budget. SFTD ensures that most of the tasks get a portion of the budget allocated, so they do not need to wait for the VMs to become idle. In both approaches, allocating resources to the entry tasks guarantees the execution of the workflow.

3.4.2 Resource Provisioning and Scheduling

Once the sub-budgets are assigned to each task, the algorithm processes entry tasks of the workflow and puts them into the queue sorted by their EFT in ascending order. Then, it schedules each task in the queue in the following ways. First, the algorithm tries to reuse an idle VM that has the input data of the task in its cache. If such a VM exists, then the task is assigned to it, always favouring VMs in which executing the task would lead to the lowest cost. Notice that the lowest cost ($0) is achieved when a VM can finish the task before its next billing period. If there is no idle VM with cached input data, then the algorithm tries to reuse any currently idle VM. If no

Algorithm 2 Resource Provisioning and Scheduling

```
procedure SCHEDULEQUEUETASKS($q$)
    sort $q$ by ascending Earliest Finish Time (EFT)
    $sb$ = spare budget
    while $q$ is not empty do
        $t = q.poll$
        $vm = null$
        $delayFlag =$ false
        if there are idle VMs then
            $VM_{idle} =$ set of all idle VMs
            $VM_{idle}^{input} =$ set of $vm \in VM_{idle}$ that have $t$'s input data
            $vm = vm \in VM_{idle}^{input}$ that can finish $t$ with minimum risk of incurring
            a new billing period
            if $vm = null$ then
                $vm = vm \in VM_{idle}$ that can finish $t$ with minimum risk of incurring
                a new billing period
        else
            $vmt =$ cheapest VM type
            if $t.budget < C_{vmt}^{t}$ then
                $delayFlag =$ true
            if $delayFlag =$ false then
                $vmt =$ fastest VM type within $t.budget$
                if there are faster VM type than $vmt$ AND $sb$ is enough then
                    $vmt = leaseFasterVMT()$
                $vm = provisionVM(vmt)$
        $scheduleTask(t, vm)$
```

Algorithm 3 Budget Update

```
procedure UPDATEBUDGET($T$)
    $t_f =$ completed task
    $T_c$ = set of $t \in T$ that are children of $t_f$
    $\beta_c$ = total sum of $t.budget$, where $t \in T_c$
    $T_u$ = set of unscheduled $t \in (T - T_c)$
    $\beta_u$ = total sum of $t.budget$, where $t \in T_u$
    $sb$ = spare budget
    if $C_{vmt}^{t_f} \leq (t_f.budget + sb)$ then
        $sb = (t_f.budget + sb) - C_{vmt}^{t_f}$
    else
        $debt = C_{vmt}^{t_f} - (t_f.budget + sb)$
        $\beta_c = \beta_c - debt$
        DISTRIBUTEBUDGET($\beta_c$, $T_c$)
        if $\beta_c < 0$ then
            $\beta_u = \beta_u + \beta_c$
            DISTRIBUTEBUDGET($\beta_u$, $T_u$)
```

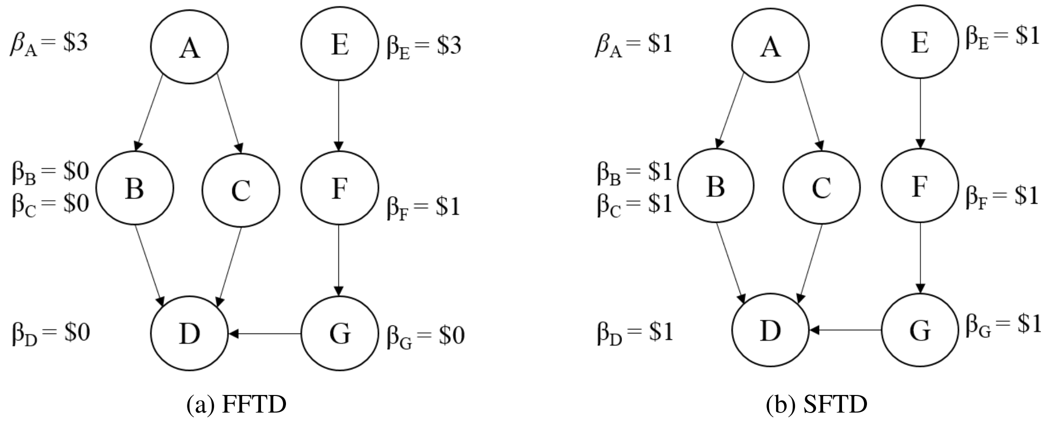


Fig. 3.2: Sample of budget distribution scenario

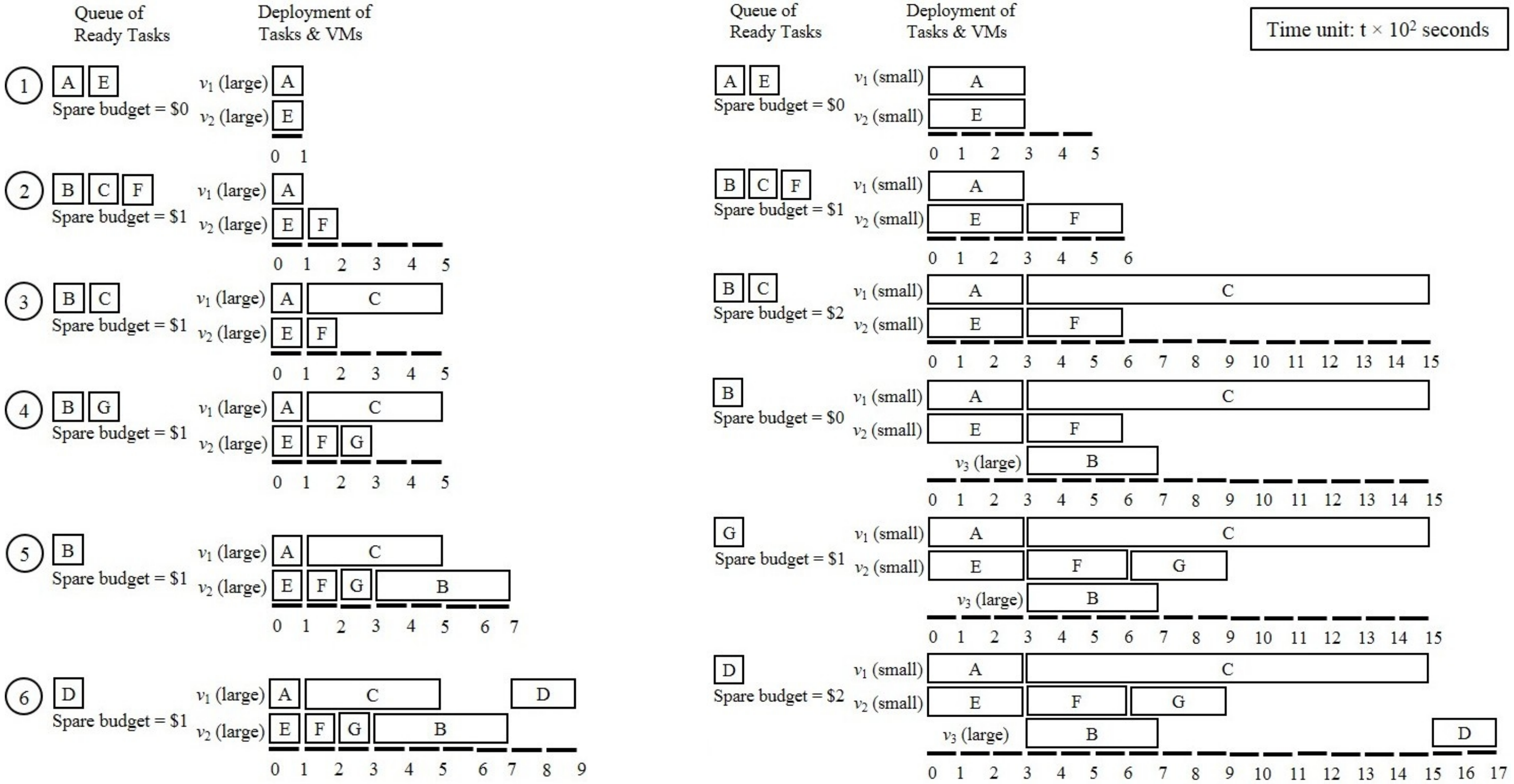
idle VMs are available, then a new VM of the fastest affordable type with the task's sub-budget is provisioned. The resource provisioning and scheduling algorithm are shown in Algorithm 2.

After a task is completed, the algorithm updates the budget distribution of all the remaining tasks to reflect the budget spent so far. Also, it maintains any unused sub-budgets of the tasks that have reused idle VMs as a spare budget. This extra budget is used either when updating the budget distribution or in the provisioning phase to lease faster VM types. The budget-updating algorithm can be seen in the Algorithm 3.

3.4.3 Illustrative Example

This section explains how the workflow shown in Fig. 3.1 would be scheduled using the proposed strategies. In this example, we assumed that PC_{vmt} of each VM type is linearly proportional to c_{vmt}. The resulting budget distribution produced by FFTD and SFTD is shown in Fig. 3.2a and Fig. 3.2b respectively and their corresponding schedules are shown in Fig. 3.3a and 3.3b. Furthermore, the *Queue of Ready Tasks* column represents the tasks that are ready for execution. In contrast, the *Deployment of Tasks & VMs* presents the deployment of the ready task to the leased VMs and the *spare budget* shown presents the values after the tasks are deployed in each step.

We can see that steps 1 to 3 for FFTD and SFTD are similar except for the VM type provisioned and the spare budget. The VM type chosen is different based on the sub-budget allocated to the entry tasks for which new VMs are provisioned. In Step 4 of FFTD, the algorithm delays task B because its sub-budget is $0. After task F finishes, task G becomes ready for execution. The algorithm prioritizes task G to be scheduled because it has lower EFT than task B. Then, task G



(a) FFTD

(b) SFTD

Fig. 3.3: Sample of scheduling and resource provisioning scenario

reuses $v_2(large)$, and task B reuses the same VM after task G finishes. Task D becomes ready for execution after tasks B and C execute and reuses $v_1(large)$. Finally, FFTD costs \$6 for the execution; it has \$1 spare budget left and obtains a makespan of 900s.

In Step 4 of SFTD, the algorithm schedules task B and provisions a new small VM based on its sub-budget. Eventually, there is enough extra budget for leasing a faster VM type. Hence, the algorithm leases a new large VM for task B. After task F completes, task G becomes ready for execution and reuses $v_2(small)$. Then, task D becomes ready for execution after tasks B and C finish and reuses $v_3(large)$, In the end, SFTD costs \$5 for the execution–it has \$2 spare budget left and produces a 1700s makespan.

3.5 Performance Evaluation

To evaluate the algorithms' performance, we used synthetic workflows created based on the characteristics of five well-known real workflow applications from different scientific areas. They were generated using the WorkflowGenerator[1] tool. The runtime estimate produced by this tool for each task was used as the size of the task instead.

The Montage (astronomy) workflow is used to produce sky mosaics from a set of input images. Most of its tasks are I/O intensive, and they do not require much CPU processing. The LIGO (astrophysics) workflow is used to detect gravitational waves. It consists of mostly CPU intensive tasks with high memory requirements. The SIPHT (bioinformatics) workflow is used for the automatic searching of sRNA encoding-genes. Most of the tasks in SIPHT have high CPU and low I/O utilization. Epigenomics (bioinformatics) is a CPU intensive workflow that is used for executing various genome-sequencing operations. Finally, the Cybershake (earthquake science) workflow generates synthetic seismograms to characterize earthquake hazards and is considered data-intensive with substantial memory and CPU requirements.

Different budget intervals were used in the experiments. We assumed that the minimum budget to run the workflow is equal to the cost of running all tasks on a single VM of the cheapest type. Based on this budget, we defined ten different budget intervals, as seen in Eq. 3.11.

$$budget = \alpha * min_{budget} \quad where \quad 0 < \alpha < 11 \tag{3.11}$$

[1] https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator

Table. 3.2: VM types and prices used

Name	**Memory (GB)**	**vCPU**	**ECU**	**Price per Hour** ($)
small	3.75	2	7	1
medium	7.5	4	14	2
large	15	8	28	4
xlarge	30	16	56	8

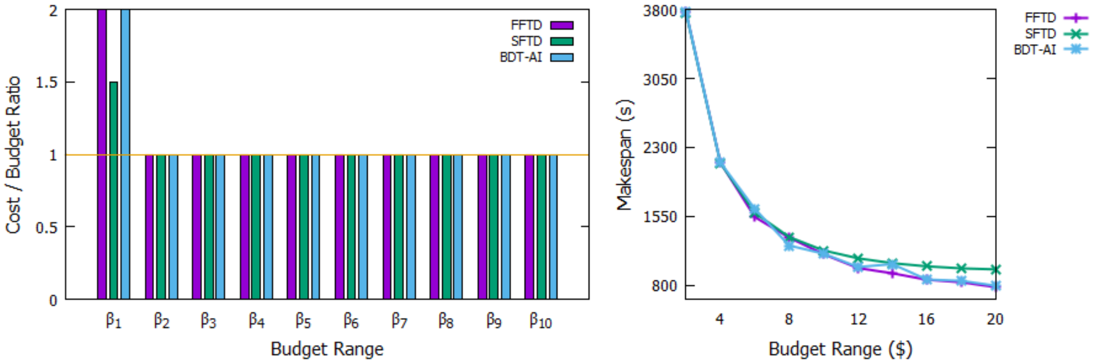
The tightest budget in the range corresponds to a budget estimated with $\alpha = 1$, while the most relaxed one was calculated using $\alpha = 10$.

We used CloudSim [72] to model an IaaS provider with a single data center and four VM types. The VM type configurations are shown in Table 3.2. Their configurations are a simplified version of the compute-optimized (c4) instance types offered by Amazon EC2 that have a linear relationship between the processing capacity and price. An hourly billing period was modelled for all VM types, and the provisioning delays were set to 97 seconds based on the study by Mao and Humphrey [38]. The CPU performance of VMs was degraded by at most 24% based on a normal distribution with a 12% mean and a 10% standard deviation, as reported by Jackson et al. [37].

3.5.1 Algorithm Performance

The goal of these experiments was to evaluate the algorithm's performance regarding cost and makespan. The cost performance was measured using the cost to budget ratio to assess the algorithm's ability to meet the budget constraint. Ratio values higher than one indicate a cost higher than the budget, values equal to one mean a cost equal to the budget, and values smaller than one represent a cost lesser than the budget. Furthermore, the experiments for each budget interval were repeated 100 times, and we plotted the mean value in the charts.

We compared our algorithm with Budget Distribution with Trickling (BDT) [107], a dynamic level-based budget distribution algorithm that has similar objectives to our solution. BDT schedules the tasks based on their Earliest Start Time (EST) and introduces a Time-Cost Trade-off Factor (TCTF), which calculates the trade-off ratio between cost and time for executing a task in a VM type. Then, it selects the VM type with the largest value in the resource provisioning phase. BDT executes all tasks in a level using the available budget and then trickles down the leftover budget to the level below. It delays tasks whenever the budget is not enough to lease new VMs and forces



(a) Cost/Budget Ratio

(b) Makespan

Fig. 3.4: Cost/budget ratio and makespan performance of Montage

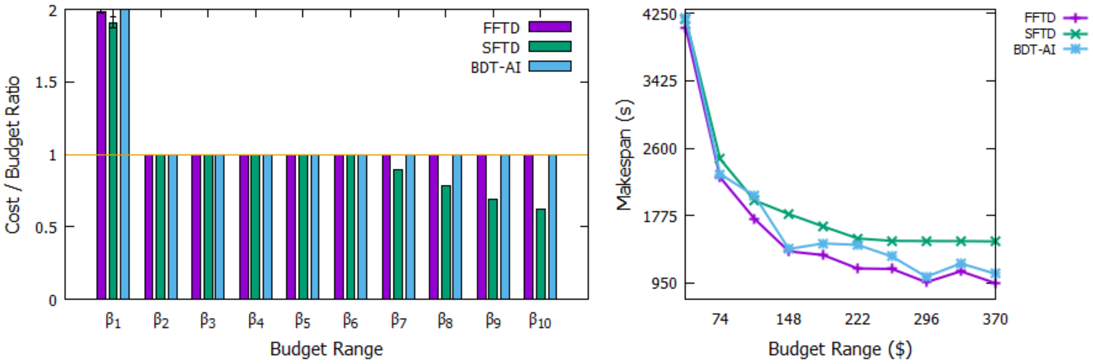
them to reuse VMs when possible. The authors of BDT introduce several budget distribution strategies, and the ‘All-In’ scenario presents the best performance. Hence, we used BDT-AI (BDT All-In), to evaluate our proposed algorithm.

Budget Constraint Evaluation

To analyze how the algorithms perform in terms of meeting the budget constraint, we plotted the cost/budget ratio values for each workflow and budget interval in Fig. 3.4a, 3.5a, 3.6a, 3.7a, and 3.8a. For the Montage workflow, the results are presented in Fig. 3.4a. SFTD performs better than the other algorithms in the strictest budget interval–this is probably due to its choice of the cheapest VM type when making provisioning decisions.

Interestingly, the performance of the algorithms is the same for the remaining intervals, with the ratio values being equal in every case. A possible reason for this is related to the coarse-grained billing period. We can see in Fig. 3.4b that the makespan obtained by all the algorithms starting from β_2 is much smaller than the length of a billing period. This result means that VM performance degradation is not significantly affecting the cost because the difference between the makespan and the billing period is wide enough to tolerate this degradation. Although there is no difference in the performance between FFTD and BDT-AI in terms of meeting the budget constraint, we found that FFTD outperforms BDT-AI in terms of the makespan–this is further discussed in the **Makespan Evaluation** subsection.

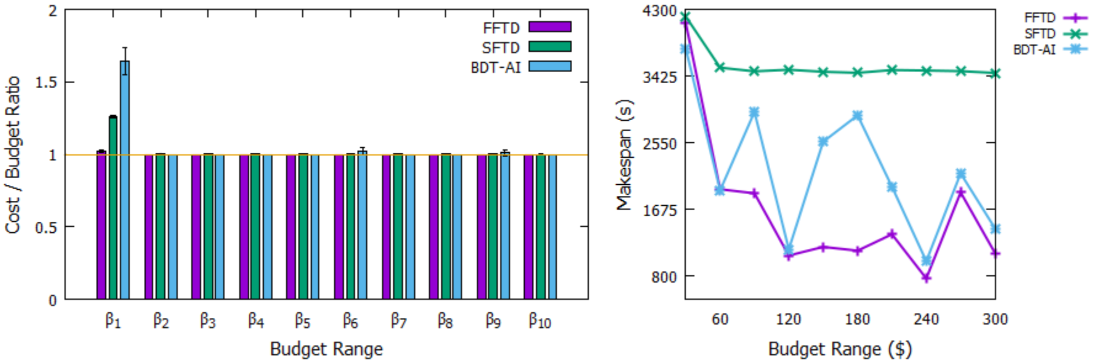
The results obtained for the LIGO workflow are shown in Fig. 3.5a. We can see that the first budget interval is too strict. Hence, all algorithms violate budget constraints. In general, SFTD



(a) Cost/Budget Ratio

(b) Makespan

Fig. 3.5: Cost/budget ratio and makespan performance of LIGO



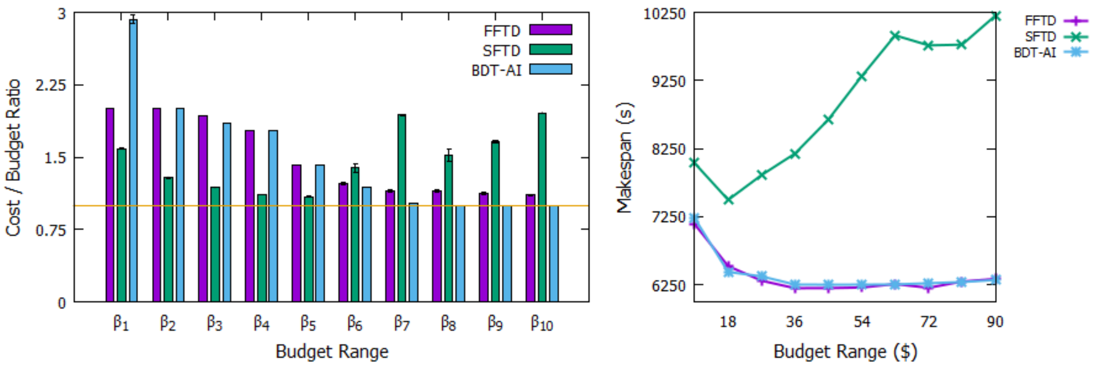
(a) Cost/Budget Ratio

(b) Makespan

Fig. 3.6: Cost/budget ratio and makespan performance of SIPHT

produces lower cost/budget ratio values even though the other two algorithms are also able to meet the budget constraint for the remaining cases. Meanwhile, Fig. 3.6a depicts the results for the SIPHT workflow. All algorithms violate the first budget interval with FFTD exceeding the budget by the smallest value. We can observe that the performance variation affects the algorithms' ability to meet the budget constraint from the marginal difference between the cost and the budget in the graphs. In general, FFTD outperforms the other two algorithms in meeting the budget constraints for the SIPHT workflow.

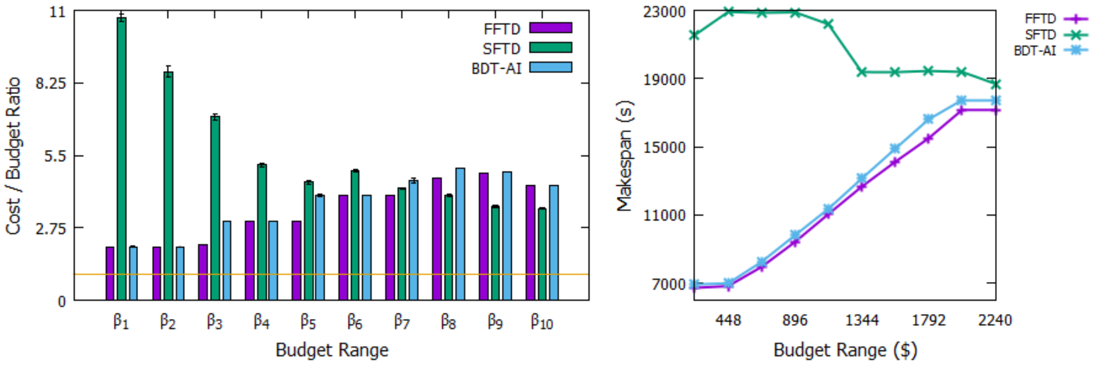
Fig. 3.7a shows the results obtained for the Cybershake workflow. In general, all algorithms fail to meet the budget constraint in every case except for BDT-AI, which succeeds in achieving its goal in the last three intervals. A possible explanation for these results is the Cybershake



(a) Cost/Budget Ratio

(b) Makespan

Fig. 3.7: Cost/budget ratio and makespan performance of Cybershake



(a) Cost/Budget Ratio

(b) Makespan

Fig. 3.8: Cost/budget ratio and makespan performance of Epigenomics

workflow characteristics as a data-intensive workflow that involves extensive I/O activities. This I/O overhead is evidence in the SFTD results–the cost/budget ratio values increase as the number of VMs increases. The larger the number of VMs is, the higher the number of files that have to be transferred over the network, and even though all algorithms consider data transfer times when estimating a task's processing time, network traffic congestion causes unpredicted costs. As a result, with SFTD being the algorithm that provisions the highest number of VMs, its schedules are profoundly impacted by network congestion. It is worthwhile mentioning that the ratio values for FFTD and BDT-AI decrease as the budget increases. While SFTD allocates more VMs as the budget increases, FFTD and BDT-AI use the additional budget to provision faster VMs instead.

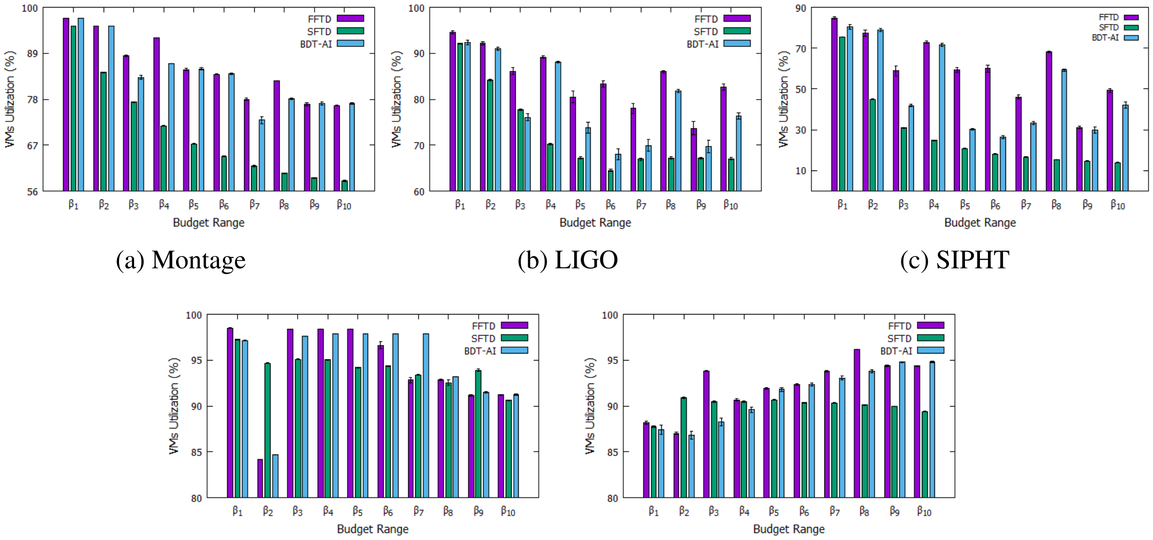
The Epigenomics workflow results are shown in Fig. 3.8a. Contrary to the Cybershake, in the Epigenomics case, SFTD leases faster VMs rather than increasing the number of leased VMs. It produces a high cost/budget ratio value for the earlier budget intervals; however, the ratios consistently decrease as the budget increases. A possible reason is that the number of VMs provisioned by SFTD remains relatively constant throughout the budget intervals. However, FFTD outperforms the other two algorithms regarding meeting the budget constraint for most of the cases.

Overall, FFTD demonstrates equal or better performance than BDT-AI in 88% of the cases regarding cost/budget ratio values. The only case where BDT-AI outperforms FFTD is for the Cybershake workflow, within which 90% of the cases BDT-AI obtains equal or better performance. However, it needs to be mentioned that the difference in performance is marginal in some cases, such as in the SIPHT workflow.

Makespan Evaluation

The first half of the budget intervals in the Montage workflow (Fig. 3.4b) shows a marginal difference in makespan. This result is due most likely to the characteristics of the tasks in the Montage workflow that highly depend on I/O rather than CPU processing (I/O-bound workflows). Hence, choosing faster VM types does not significantly affect the makespan. However, as the budget increases, the second half of the budget intervals lead to a larger makespan difference. A possible reason is due to a large number of VMs provisioned by SFTD at runtime that contributes to the communication overhead. Finally, FFTD obtains lower makespan than BDT-AI in 70% of the cases in which both algorithms achieve the same performance regarding cost/budget ratio.

Fig. 3.5b depicts the results obtained for the LIGO workflow. Although the graph's trend is similar to the Montage results, the difference between all algorithms is more clearly observed in this case. FFTD shows the lowest makespan for all cases. The results obtained for the SIPHT workflow are depicted in Fig. 3.6b. Similar to the previous results, FFTD obtains the lowest makespan and displays more stable results than BDT-AI. Morover, the results for the Cybershake workflow are shown in Fig. 3.7b. FFTD produces slightly lower makespans in 60% of the cases when compared to BDT-AI, while SFTD performs the worst. Cybershake is considered as a data-intensive workflow that involves a high number of data transfer activities. This degradation explains SFTD's performance as it provisions a larger number of VMs as the budget increases.



(a) Montage

(b) LIGO

(c) SIPHT

(d) Cybershake

(e) Epigenomics

Fig. 3.9: VMs utilization for different workflow applications

The Epigenomics workflow results are shown in Fig. 3.8b. The makespan trend is different from the other scenarios in which both FFTD and BDT-AI have a larger makespan as the budget increases. A possible explanation is a fact that Epigenomics consists of tasks that are both CPU and I/O intensive. Having a small number of VMs provisioned is probably the best approach for executing this workflow. It needs to be noted that this behaviour should be a guide for the users when defining the budget for Epigenomics workflow. Finally, FFTD shows the lowest makespan in all scenarios.

Overall scenario, FFTD demonstrates lower makespan in 84% of all cases. In the case where FFTD gets an equal performance to BDT-AI in terms of meeting the budget constraint, it obtains a lower makespan in 80% of the cases. Meanwhile, in the cases in which FFTD achieves lower cost/budget ratios, which is usually accompanied by higher makespans, it also successfully obtains lower makespans than BDT-AI in 93% of the cases. Nevertheless, in some cases, the difference in makespan is marginal.

3.5.2 VM Utilization

To better understand the behaviour of the algorithms, we analyzed the average VM utilization for each workflow. High VM utilization means the algorithm is capable of mapping tasks to VMs efficiently by utilizing idle time slots. Hence, this performance metric is suitable to evaluate the algorithms' ability to deal with coarse-grained IaaS cloud billing periods. The VMs utilization results are shown in Fig. 3.9.

For the Montage workflow, FFTD presents higher VM utilization in 50% of the cases than BDT-AI. Meanwhile, in the LIGO workflow scenario, FFTD achieves the highest VM utilization in all cases. On average, the SIPHT workflow shows the lowest VM utilization for all of the experiments. However, FFTD obtains a higher VM utilization in 90% of the cases when compared to BDT-AI for SIPHT. Furthermore, BDT-AI presents a higher VM utilization in 60% of the cases than FFTD for the Cybershake workflow–this supports the cost/budget ratio and makespan results. Finally, in the Epigenomics workflow, FFTD produces higher VM utilization than BDT-AI in 80% of the cases.

Overall, in 72% of the cases, FFTD demonstrates better performance than BDT-AI regarding VM utilization. However, for the Cybershake workflow, BDT-AI performs better than FFTD. A possible explanation is because the level-based strategy of BDT-AI works best for data-intensive workflows that have a high degree of parallelism.

3.6 Summary

In this chapter, we presented a task-based budget distribution strategy for executing scientific workflows in IaaS clouds with coarse-grained billing periods. The problem was modelled as a workflow resource provisioning and scheduling problem, which aims to minimize the makespan while meeting the user-defined budget. Furthermore, the proposed strategy exploits the independent task readiness for executing the workflow.

The algorithm distributes the workflow budget to each task and drives the resource usage through the sub-budget of each task. It schedules tasks individually based on their earliest finish time whenever their parent tasks have completed execution, and their input data are available. The algorithm implements a VM reusing policy to utilize the idle time slots that occur due to the coarse-grained billing periods. It provisions the fastest VM type possible within the budget whenever it is necessary due to the unavailability of reusable idle VMs. Every time a task finishes, the algorithm considers the budget spent so far and adjusts the next task scheduling decision if necessary. For each task that reuses idle VMs, its unused sub-budget is kept as a spare budget and utilized to update the budget distribution or to lease faster VMs in the resource provisioning phase.

The performance evaluation results demonstrate that our solution has an overall better performance than the state-of-the-art algorithm. It successfully obtains 88% equal or better performance regarding cost/budget ratio values and achieves lower makespans in 84% of the cases. It gets higher VM utilizations in 72% of the experiments. In the next chapter, we investigate this approach on dynamic workloads of multiple workflows with finer-grained billing periods (i.e., seconds). In those scenarios, there is an opportunity to enhance the scheduling processes that can be achieved from the sharing and reusing VMs between different workflows.

This page intentionally left blank.

Chapter 4

A Budget-constrained Scheduling Algorithm for Multiple Workflows

In this chapter, we propose a resource-sharing policy to improve system utilization and to fulfil various QoS requirements from multiple users in WaaS platforms. We develop an ***E****lastic* ***B****udget-constrained resource* ***P****rovisioning and* ***S****cheduling algorithm for* ***M****ultiple workflows (EBPSM) that can reduce the computational overhead by enforcing resource sharing to minimize workflows' makespan while meeting a user-defined budget. Our experiments show that the EBPSM algorithm can utilize the resource-sharing policy to achieve higher performance in terms of minimizing the makespan compared to the state-of-the-art budget-constraint scheduling algorithm.*

4.1 Introduction

Designing algorithms for scheduling scientific workflow executions in clouds is not trivial. The problems have attracted many computer scientists into cloud workflow scheduling research to fully utilize the clouds' capabilities for efficient scientific workflows execution [110] [26]. The majority of those studies focus on the scheduling of a single workflow in cloud computing environments. In this model, they assume an individual user utilizes a WMS to execute a particular workflow's job in the cloud. The WMS manages the execution of the workflow so that it can be completed within the defined QoS requirements. Along with the growing trend of scientific workflows adoption in the community, there is a need for platforms that provide workflow execution as a service.

To fulfil the users' various QoS requirements, a traditional WMS may process the workflows individually in a dedicated set of VMs, as discussed in Chapter 1 in Fig. 1.3a. This approach, after all, is the simplest way to ensure the QoS fulfilment of each job. In this scenario, a WMS manages

This chapter is derived from: **Muhammad H. Hilman**, Maria A. Rodriguez, and Rajkumar Buyya. 'Resource-sharing Policy in Multi-tenant Scientific Workflow-as-a-Service Platform.' *Journal of Computer and System Sciences (JCSS)*, 2020 (under review).

different types of tasks' execution by tailoring their specific software configurations and requirements to a VM image. The VM containing this image can then be quickly deployed whenever a particular workflow is submitted. However, this model cannot easily be implemented in WaaS platforms, where many users with different workflow applications are involved. We cannot naively simplify the assumption where every VM image can be shared between multiple users with different requirements. Multiple workflow applications may need different software configurations, which implies a possible dependency conflict if they are fitted within a VM image. This assumption also creates a more complex situation where, at any given time, a new workflow application type needs to be deployed. This newly submitted job cannot reuse the already provisioned VMs as they may not contain its software configurations.

Therefore, adopting an appropriate resource-sharing policy and, at the same time scheduling multiple workflows simultaneously, as explained in Chapter 1 in Fig. 1.3b, is considerably preferred for WaaS platforms. We argue that introducing this strategy creates a more efficient platform as it reduces the unnecessary overhead during the execution. The efficiency may be gained from sharing the same workflow application software for different users by tailoring a specific configuration in a container image instead of a VM image. This strategy enables (i) sharing and reusing already provisioned VMs between users to utilize the inevitable scheduling gaps from intra-workflow's dependency and (ii) sharing local cached images and datasets within a VM that creates a locality, which eliminates the need for network activities before the execution.

Based on these problems and requirements, we propose EBPSM, an **E**lastic **B**udget-constrained resource **P**rovisioning and **S**cheduling algorithm for **M**ultiple workflows designed for WaaS platforms. An elaboration to the resource sharing policy that has been introduced in [80] by further utilizing container technology and VM local storage to share datasets and computational resources. This proposed algorithm focused more on a budget-constrained scheduling scenario where its budget distribution strategy has been discussed in Chapter 3.

The EBPSM algorithm can make a quick decision to schedule the workflow tasks dynamically and empower the sharing of software configuration and reuse of already provisioned VMs between users using container technology. Our algorithm also considered the inherent features of clouds that affect multiple workflows scheduling, such as performance variation of VMs [4] and provisioning delay [39] into the VMs auto-scaling policy. Furthermore, EBPSM implemented an efficient budget distribution strategy that allows the algorithm to provision the fastest VMs possible. This is meant to minimize the makespan and adopt the container images and datasets sharing policy to eliminate the need for network transfer between tasks' execution. Our extensive experiments showed that EBPSM, which utilizes the resource-sharing system, can significantly reduce the overhead. This, in turn, implies the minimization of workflows' makespan.

The rest of this chapter is organized as follows: Section 4.2 reviews works that are related to this proposal. Section 4.3 explains the problem formulation of multiple workflow scheduling in WaaS platforms, including the assumption of application and resource models. The proposed algorithm is described in Section 4.4 followed by the performance evaluation in Section 4.5. Finally, the Section 4.6 summarizes the findings.

4.2 Related Work

The majority of works in multiple workflows scheduling pointed out the necessity of reusing already provisioned VMs to reduce the idle gaps and increase system utilization. Examples include the CWSA [17] that uses a depth-first search technique to find potential schedule gaps between tasks' execution. Another work is the CERSA [79] that dynamically adjusts the VM allocation for tasks in a reactive fashion whenever a new workflow job is submitted to the system. These works' idea to fill the schedule gaps between tasks' execution of a workflow to be utilized for scheduling tasks from another workflow is similar to our proposal. However, they assume that different workflow applications could be deployed into any existing VM available without considering the possible complexity of software dependency conflicts. Our work differs in the way that we model the software configurations into a container image before deploying it to the VMs for execution.

The use of the container for deploying scientific workflows has been intensively researched. Examples include the work of Qasha et al. [111] that deployed a TOSCA-based workflow[1] using Docker[2] container on e-Science Central platform[3]. Although their work is done on a single VM, the result shows a promising future scientific workflows reproducibility using container technology. A similar result is presented by Liu et al. [112] that convinces less overhead performance and high flexibility of deploying scientific workflows using Docker containers. Finally, the adCFS

[1] https://github.com/ditrit/workflows

[2] https://www.docker.com/

[3] https://www.esciencecentral.org/

[113] is designed to schedule containerized scientific workflows that encourage a CPU-sharing policy using a Markov-chain model to assign the appropriate CPU weight to containers. Those solutions are the early development of containerized scientific workflows execution. Their results show high feasibility to utilize container technology for efficiently bundling software configurations for workflows that are being proposed for WaaS platforms.

One of the challenges of executing scientific workflows in the clouds is related to the data locality. The communication overhead for transferring the data between tasks' execution may take a considerable amount of time that might impact the overall makespan. A work by Stavrinides and Karatza [61] shows that the use of distributed in-memory storage to store the datasets locally for tasks' execution can reduce the communication overhead. Our work is similar regarding the data locality policy to minimize the data transfer. However, we propose the use of cached datasets in VMs local storage to endorse the locality of data. We enhance this policy so that the algorithm can intelligently decide which task to be scheduled in specific VMs that can provide the minimum execution time given the available cached datasets.

Two conflicting QoS requirements in scheduling (e.g., time and cost) have been a significant concern when deploying scientific workflows in clouds. A more relaxed constraint to minimize the trade-off between these requirements is shown in several works that consider scheduling workflows within the deadline and budget constraints. They do not attempt to optimize one or both of the QoS requirements, but instead maximizing the success rate of workflows execution within the constraints. Examples of these works include MW-DBS [55] and MW-HBDCS [94] algorithms. Another similar work is the MQ-PAS [56] algorithm that emphasizes on increasing the providers' profit by exploiting the budget constraint as long as the deadline is not violated. Our work considers the user-defined budget constraint. However, it differs in the way that the algorithm aims to optimize the overall makespan of workflows while meeting their budget.

Several works specifically focus on handling the real-time workload of workflows in WaaS platforms. This type of workload raises uncertainty issues, since the platforms have no knowledge of the arriving workflows. EDPRS [71] adopts a dynamic scheduling approach using event-driven and periodic rolling strategies to handle the uncertainties in real-time workloads. Another work, called ROSA [78], controls the queued jobs–which increase the uncertainties along with the performance variation of cloud resources–in the WaaS platforms to reduce the waiting time that can prohibit the uncertainties propagation. Both algorithms are designed to schedule multiple workflows dynamically to minimize the operational cost while meeting the deadline. Our algorithm has the same purpose of handling the real-time workload and reducing the effect of uncertainties in WaaS platforms. However, we differ in that our scheduling objectives are minimizing the workflows' makespan while meeting the user-defined budget.

The majority of works in workflow scheduling that aim to minimize the makespan, while meeting the budget constraints, adopt a static scheduling approach. This approach finds a near-optimal solution of mapping the tasks to VMs–with various VM types configuration–to get a schedule plan before runtime. Examples of these works include MinMRW-MC [114], HEFT-Budg, and MinMin-Budg [115]. However, this static approach is considered inefficient for WaaS platforms as it might increase the waiting time of arriving workflows due to the intensive pre-processing computation time to generate a schedule plan.

On the other hand, several works consider the available user-defined budget dynamically drives scheduling budget-constrained workflows. In this area, examples include BAT [116], which distributes the budget of a particular workflow to its tasks by trickling down the available budget based on the depth of tasks. Another work, MSLBL [117] algorithm allocates the budget by calculating a proportion of the sub-budget efficiently to reduce the unused budget. However, those solutions are designed for a single cloud workflow scheduling scenario. To the best of our knowledge, none of these types of budget-constrained algorithms that aim to tackle the problem in multiple workflows–which resembles the problem in WaaS platforms–has been proposed.

4.3 Application and Resource Model

We considered a workload of workflows that are modelled as DAGs (Directed Acyclic Graphs). Each workflow w is associated with a budget β_w that is defined as a soft constraint of cost representing users' willingness to pay for the execution of the workflows.

The task t is executed within a container–that bundles software configurations for a workflow in a container image–which is then deployed on VMs. A container provisioning delays $prov_c$ is acknowledged to download the image, setup, and initiate the container on an active VM. In this chapter, we set aside the idea of co-locating several running containers within a VM for further study, due to the focus of the work is to explore the sharing policy of VMs in terms of its computing, storage, and network capacities. Therefore, we assumed that only one container could run on top of a VM at a particular time. Therefore, the same host VM performance of CPU, memory, and bandwidth can be achieved for a container. Once the container is deployed, VM local storage maintains the images, so it can be reused without the need to re-download them. We assumed the scheduler sends custom signals by using commands in the container to trigger tasks' execution within containers to avoid the necessity of container redeployment.

We considered a pay-as-you-go scheme in IaaS clouds, where VMs are provisioned on-demand and are priced per billing period bp (i.e., per-second, per-minute, per-hour). Hence, any partial use of the VM is rounded up and charged based on the nearest bp. In this chapter, we assumed a fine-grained per-second bp because it is lately being adopted by the majority of IaaS cloud providers including Amazon EC2[4], Google Cloud[5], and Azure[6]. We modelled a data center within a particular availability zone from a single IaaS cloud provider to reduce the network overhead and eliminate the cost associated with data transfer between zones. Our work considers a heterogeneous cloud environment model where VMs with different VM types $VMT = \{vmt_1, vmt_2, vmt_3, ..., vmt_n\}$ which have various processing power p_{vmt} and different cost per billing period c_{vmt} can be leased. We considered that all types of VM always have an adequate memory capacity to execute the various type of workflows' tasks. Finally, we could safely assume that the VM type with a higher p_{vmt} has more expensive c_{vmt} than the less powerful and slower ones.

Each VM has a bandwidth b_{vmt} that is relatively the same between different VM types as they come from a single availability zone. We did not restrict the number of VMs to be provisioned during the execution, but we also acknowledged the delay in acquiring VMs $prov_{vmt}$ from the IaaS provider. We assumed that the VMs could be eliminated immediately from the WaaS platform without additional delay. Furthermore, we considered the performance variation of VMs that might come from a virtualized backbone technology of clouds, geographical distribution, and multi-tenancy [4] and that CPU of VM advertised by IaaS provider is the highest CPU capacity that can be achieved by the VMs. We did not assume another degradation of using containerized environments because Kozhirbayev and Sinnott [118] have reported its near-native performance.

[4] https://aws.amazon.com/about-aws/whats-new/2017/10/announcing-amazon-ec2-per-second-billing/

[5] https://cloud.google.com/blog/products/gcp/extending-per-second-billing-in-google

[6] https://azure.microsoft.com/en-us/pricing/details/virtual-machines/windows/

A global storage system GS is modelled for data sharing between tasks (e.g., Amazon S3) with unlimited storage capacity. This global storage has reading rates GS_r and writing rates GS_w respectively. In this model, the tasks transfer their outputs from the VMs to the storage and retrieve their inputs from the same place before the execution. Therefore, the network overhead is inevitable and is one of the uncertainties in clouds as the network performance degradation can be observed due to the amount of traffic and the virtualized backbone [119]. To reduce the need for accessing storage GS for retrieving the data, VMs local storage LS_{vmt} is also modelled to maintain d_t^{in} and d_t^{out} after particular tasks' execution using FIFO policy. It means the earliest stored data will be deleted whenever the capacity of LS_{vmt} cannot accommodate new data needing to be cached. Furthermore, the time taken to retrieve the input data for a particular task's execution from global storage is shown in Eq. 4.1.

$$T_{vmt}^{d_t^{in}} = (d_t^{in} / b_{vmt}) + (d_t^{in} / GS_r) \tag{4.1}$$

It is worth noting that there is no need to transfer the input data from the global storage whenever it is available in the VM as cached data from previous tasks execution. Similarly, the time needed for storing the output data in the storage is depicted in Eq. 4.2.

$$T_{vmt}^{d_t^{out}} = (d_t^{out} / b_{vmt}) + (d_t^{out} / GS_w) \tag{4.2}$$

The runtime RT_{vmt}^t of a task t in a VM of type vmt is assumed available to the scheduler as part of the scheduling process. The fact is that this runtime can be estimated using various approaches including machine learning techniques [36], but we simplified the assumption where it is calculated based on the task's size S_t in Million Instructions (MI) and the processing capacity p_{vmt} of the particular VM type in Million Instructions Per Second (MIPS) as shown in Eq. 4.3.

$$RT_{vmt}^t = S_t / p_{vmt} \tag{4.3}$$

It needs to be noted that this RT_{vmt}^t value is only an estimate and the scheduler does not depend on it being 100% accurate as it represents one of the uncertainties in clouds. Furthermore, a maximum processing time of a task in a VM type PT_{vmt}^t consists of reading the input data required from the storage, executing the task, and writing the output to the storage which are depicted in Eq. 4.4.

$$PT^{t}_{vmt} = T^{d^{in}_{t}}_{vmt} + RT^{t}_{vmt} + T^{d^{out}_{t}}_{vmt} \tag{4.4}$$

From the previous equations, we calculated the maximum cost C^{t}_{vmt} of executing a task t on a particular *vmt* as shown in Eq. 4.5.

$$C^{t}_{vmt} = \lceil (prov_{vmt} + prov_{c} + PT^{t}_{vmt})/bp \rceil * c_{vmt} \tag{4.5}$$

The budget-constrained scheduling problem that is being addressed in this paper is concerned with minimizing the makespan (i.e., actual finish time of t_{exit}) of workflow while meeting the user-defined budget as depicted in Eq. 4.6.

$$min \quad AFT(t_{exit}) \quad \text{while} \quad \sum_{n=1}^{T} C^{t_n}_{vmt} \leq \beta_w \tag{4.6}$$

Intuitively, the budget β_w will be spent efficiently on PT^{t}_{vmt} if the overhead components $prov_{vmt}$ and $prov_{c}$ that burden the cost of a task's execution can be discarded. This implies to the minimization of $AFT(t_{exit})$ as the fastest VMs can be leased based on the available budget β_w. Another important note is that, further minimization can be achieved when the task t is allocated to the VM with d^{in}_{t} available, so the need for $T^{d^{in}_{t}}_{vmt}$ which is related to the network factor that becomes one of the sources of uncertainties can be eliminated.

However, it needs to be noted that there exist some uncertainties in RT^{t}_{vmt} as the estimate of S_t is not entirely accurate, and the performance of VMs PC_{vmt} can be degraded at any time. Hence, there must be a control mechanism to ensure that these uncertainties do not propagate throughout the tasks' execution and cause a violation of β_w. This control can be done by evaluating the real value of a task's execution cost C^{t}_{vmt} right after the task is completed. In this way, its successors can adjust their sub-budget allocation so that the total cost will not violate the budget β_w.

4.4 The EBPSM Algorithm

In this chapter, we propose a dynamic heuristic resource provisioning and scheduling algorithm designed for WaaS platforms to minimize the makespan while meeting the budget. The algorithm is developed to efficiently schedule scientific workflows in multi-tenant platforms that deal with dynamic workload heterogeneity, the potential of resource inefficiency, and uncertainties of overheads along with performance variations. Overall, the algorithm enhances the reuse of software configurations, computational resources, and datasets to reduce the overheads that become one of the critical uncertainties in cloud environments. This policy implements a resource-sharing model by utilizing container technology and VMs local storage in the decision-making process to schedule tasks and auto-scale the resources.

When a workflow is submitted to the WaaS portal, its owner may define the software requirements by creating a new container image or choosing an existing template. Whenever the user selects the existing images, the platform will identify the possibly relevant information that is maintained from the previous workflows' execution, including analyzing previous actual runtime execution and its related datasets. Furthermore, the user then defines the budget β_w that is highly likely different from various users submitting the same type of workflow.

Next, the workflow is forwarded to the WaaS scheduler and is preprocessed to assign sub-budget for each task based on the user-defined β_w. This sub-budget along with the possible sharing of software configurations and datasets will lead the decisions made at runtime to schedule a task onto either an existing VM in the resource pool or a new VM from the cloud providers. The first phase of the budget distribution algorithm is to estimate the potential tasks' execution order within a workflow. The entry task(s) in the first level of a workflow is scheduled first, followed by their children until it reaches the exit task(s). In this case, we assign every task to a level within a workflow based on the Deadline Top Level (DTL) approach, as seen in Eq. 4.7

$$level(t) = \begin{cases} 0 & \text{if} \quad pred(t) = \emptyset \\ \max\limits_{p \in pred(t)} level(p) + 1 & \text{otherwise} \end{cases} \tag{4.7}$$

Furthermore, to determine the tasks' priority within a level, we sort them based on their Earliest Finish Time (EFT) in an ascending order as shown in Eq. 4.8

$$eft(t) = \begin{cases} PT^t_{vmt} & \text{if} \quad pred(t) = \emptyset \\ \max\limits_{p \in pred(t)} eft(p) + PT^t_{vmt} & \text{otherwise} \end{cases} \tag{4.8}$$

After estimating the tasks' execution order, the algorithm iterates over the tasks and distributes the budget. This budget distribution algorithm estimates the sub-budget of a task based on the cost c_{vmt} of particular VM types. At first, the algorithm chooses VMs with the cheapest types for the task. Whenever there is any extra budget left after all tasks get their allocated sub-budgets, the algorithm uses this extra budget to upgrade the sub-budget allocation for a faster VM type starting from the earliest tasks in the order. This approach is called the Slowest First Task-based Budget Distribution (SFTD) and its details is described in Chapter 3 in Algorithm 1.

Once a workflow is pre-processed, and its budget is distributed, the scheduler begins the scheduling process, this step is illustrated in Algorithm 4. The primary objective of this algorithm is to reuse the VMs that have datasets and containers–with software configurations available–in VMs local storage that may significantly reduce the overhead of retrieving the particular input data and container images from GS. This way, the algorithm avoids the provisioning of new VMs as much as possible, which reduces the VM provisioning delay and minimizes the network overhead from data transfer and downloading container images that contribute to the uncertainties.

Furthermore, the WaaS scheduler releases all entry tasks of multiple workflows into the queue. As the tasks' execution proceeds, the child tasks–which parents are completed–become ready for execution and are released into the queue. As a result, at any point in time, the queue contains all tasks from different workflows submitted to the WaaS platform that is ready for execution. The queue is periodically being updated whenever one of the two events triggered the scheduling cycle, the arrival of a new workflow's job, and the completion of a task's execution.

In every scheduling cycle, each task in the queue is processed as follows. The first step is to find a set of VM_{idle} on the system that can finish the task's execution with the fastest time within its budget. The algorithm estimates the execution time by not only calculating PT^{t}_{vmt} but also considering possible overhead $prov_c$ caused by the need for initiating a $container$ in case the VM_{idle} does not have a suitable $container$ deployed.

At first attempt, VM_{idle} with input datasets VM^{input}_{idle} are preferred. The VM^{input}_{idle} that has the datasets available in its local storage must also cache the $container$ image from the previous execution. In this way, two uncertain factors $T^{d^{in}_{t}}_{vmt}$ and $prov_c$ are eliminated. In this case, the sub-budget for this particular task can be spent well on using the fastest VM type to minimize its execution time. This scenario is always preferred since the retrieval of datasets from GS and downloading

Algorithm 4 Scheduling

```
procedure SCHEDULEQUEUEDTASKS($q$)
    sort $q$ by ascending Earliest Finish Time (EFT)
    while $q$ is not empty do
        $t = q.poll$
        $container = t.container$
        $vm = null$
        if there are idle VMs then
            $VM_{idle}$ = set of all idle VMs
            $VM_{idle}^{input}$ = set of $vm \in VM_{idle}$ that have $t$'s input data
            $vm = vm \in VM_{idle}^{input}$ that can finish $t$ within $t.budget$ with the fastest execution time
            if $vm = null$ then
                $VM_{idle} = VM_{idle} \setminus VM_{idle}^{input}$
                $VM_{idle}^{container}$ = set of $vm \in VM_{idle}$ that have $container$ deployed
                $vm = vm \in VM_{idle}^{container}$ that can finish $t$ within $t.budget$ with the fastest execution time
                if $vm = null$ then
                    $VM_{idle} = VM_{idle} \setminus VM_{idle}^{container}$
                    $vm = vm \in VM_{idle}$ that can finish $t$ within $t.budget$ with the fastest execution time
        else
            $vmt$ = fastest VM type within $t.budget$
            $vm = provisionVM(vmt)$
        if $vm\ != null$ then
            if $vm.container\ != container$ then
                $deployContainer(vm, container)$
            $scheduleTask(t, vm)$
```

Algorithm 5 Resource Provisioning

```
procedure MANAGERESOURCE
    $VM_{idle}$ = all leased VMs that are currently idle
    $threshold_{idle}$ = idle time threshold
    for each $vm_{idle} \in VM_{idle}$ do
        $t_{idle}$ = idle time of $vm$
        if $t_{idle} \geq threshold_{idle}$ then
            terminate $vm_{idle}$
```

the *container* images through networks has become a well-known overhead that poses a significant uncertainty as its performance also may be degraded over time [119].

If VM has not been found in VM_{idle}^{input}, the algorithm finds a set of VM_{idle} that have *container* deployed. This set of $VM_{idle}^{container}$ may not have the input data available as they may have been cleared from VMs local storage. If the set still does not contain the preferred VM, any VM from the remaining set of VM_{idle} is chosen. In the latter scenario, the overhead of provisioning delay $prov_{vm}$ can be eliminated. It is still better than having to acquire a new VM. Whenever a suitable VM in the resource pool is found, the task then is immediately scheduled on it. If reusing an existing VM is not possible, the algorithm provisions a new VM of the fastest type that can finish the task within its sub-budget. This approach is the utmost option to schedule a task.

For better adaptation to uncertainties that come from performance variation and unexpected delays during execution, there is a control mechanism within the algorithm to adjust sub-budget allocations whenever a task finishes. This mechanism defines a spare budget variable that stores the residual sub-budget calculated from the actual cost execution. Whenever a task is completed, the algorithm calculates the actual cost of execution using the Eq. 4.5 and redistributes the left-over sub-budget to the unscheduled tasks. If the actual cost exceeds the allocated sub-budget, the shortfall will be taken from the sub-budget of unscheduled tasks. Therefore, the budget redistribution will take place every time a task is finished. In this way, the uncertainties (e.g., performance variation, overhead delays) that occur to a particular task do not propagate throughout the rest of the following tasks. The details of this budget update are depicted in Chapter 3 in Algorithm 3.

Regarding the resource provisioning strategy, the algorithm encourages a minimum number of VMs usage by reusing as much as possible existing VMs. The new VMs are only acquired whenever idle VMs are not available due to the high density of the workload to be processed. In this way, the VM provisioning delays overhead can be reduced. As for the deprovisioning strategy, all of the running VMs are monitored every $prov_{int}$, and all VMs that have been idle for more than $threshold_{idle}$ time are terminated as seen in Algorithm 5. The decision to keep or terminate a particular VM should be carefully considered as the cached data within its local storage is one of the valued factors that impact the performance of this algorithm. Therefore, the $prov_{int}$ and $threshold_{idle}$ are configurable parameters that can lead to a trade-off between performance in terms of resource utilization and makespan along with the VMs local storage caching policy.

4.5 Performance Evaluation

To evaluate our proposal, we used five synthetic workflows derived from the profiling of workflows [27] from various scientific areas generated using WorkflowGenerator tool[7]. The first workflow is

[7] https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator

Table. 4.1: Characteristics of synthetic workflows

Workflow	**Parallel Tasks**	**CPU Hours**	**I/O Requirements**	**Peak Memory**
Cybershake	Very High	Very High	Very High	Very High
Epigenome	Medium	Low	Medium	Medium
LIGO	Medium High	Medium	High	High
Montage	High	Low	High	Low
SIPHT	Low	Low	Low	Medium

Cybershake that makes synthetic seismograms to differentiate various earthquakes hazards. This earth-science workflow is data-intensive with very high CPU and memory requirements. The second workflow is Epigenome, a bioinformatics application with CPU-intensive tasks for executing genome-sequencing related research. The third workflow is an astrophysics application called Inspiral–part of the LIGO project–that is used to analyze the data from gravitational waves detection. This workflow consists of CPU-intensive tasks and requires a high memory capacity. The next workflow is Montage. This workflow is an astronomy application used to produce a sky mosaics image from several different angles on sky observation images. Most of the Montage tasks are considered I/O intensive while involving less CPU. Finally, we included a bioinformatics application to encode sRNA genes called SIPHT. Its tasks have relatively low I/O utilization with medium memory requirements. The resume of these workflows can be seen in Table 4.1.

The experiments were conducted with various workloads composed of a combination of workflows mentioned above in three different sizes: approximately 50 tasks (small), 100 tasks (medium), and 1000 tasks (large). Each workload contains a different number and various types of workflows that were randomly selected based on a uniform distribution, and the arrival rate was modelled based on a Poisson distribution. Every workflow in a workload was assigned a budget that is always assumed sufficient. Budget insufficiency can be managed by rejecting the job from the platform or re-negotiating the budget with the users. This budget was randomly generated based on a uniform distribution from a range of minimum and maximum cost of executing the workflow. The minimum cost was estimated from simulating the execution of all of its tasks in sequential order on the cheapest VM type. On the other hand, the maximum cost was estimated based on the parallel execution of each task on multiple VMs. In this experiment, we used the runtime generated from the WorkflowGenerator for the task's size measurement.

We extended CloudSim [72] to support the simulation of WaaS platforms. Using CloudSim, we modelled a single IaaS cloud provider that offers a data center within a single availability zone

Table. 4.2: Configuration of VM types used in evaluation

Name	vCPU (MIPS)	Storage (GB)	Price per second (cent)
Small	2	20	1
Medium	4	40	2
Large	8	80	4
XLarge	16	160	8

with four VM types that are shown in Table 4.2. These four VM types configurations are based on the compute-optimized (c4) instance types offered by the Amazon EC2, where the CPU capacity has a linear relationship with its respective price. We modelled the per-second billing period for leasing the VMs, and for all VM types, we set the provisioning delay to 45 seconds based on the latest study by Ullrich et al. [120]. On the other side, using the model published by Piraghaj et al. [121], the container provisioning delay was set to 10 seconds based on the average container size of 600 MB, a bandwidth 500 Mbps, and a 0.4 seconds delay in container initialization.

The CPU and network performance variation were modelled based on the findings by Leitner and Cito [4]. The CPU performance was degraded by a maximum of 24% of its published capacity based on a normal distribution with a 12% mean and a 10% standard deviation. Furthermore, the bandwidth available for a data transfer was potentially degraded by at most 19% based on a normal distribution with a 9.5% mean and a 5% standard deviation. In this experiment, as mentioned earlier, each VM was modelled to maintain an LS_{vmt} that stores the cached data produced during the execution based on FIFO policy. The design for a more sophisticated strategy to retain or terminate the lifetime of cached datasets within an LS_{vmt} is left for future work.

To create a baseline for EBPSM, we extended the MSLBL [117] algorithm for multiple workflows scheduling (MSLBL_MW). MSLBL was designed for single workflow execution, so we added a function to handle multiple workflows by creating a pool of arriving workflows where the algorithm then dispatched the ready tasks from all workflows for scheduling. Furthermore, MSLBL assumed that a set of computational resources are available in a fixed quantity all over the scheduling time. Therefore, to cope with a dynamic environment in WaaS platforms, we added a simple dynamic provisioner for MSLBL_MW that provisions a new VM whenever there is no existing VMs available. This newly provisioned VM is selected based on the fastest VM type that can be afforded by the sub-budget of a particular task. This dynamic provisioner also automatically terminates any idle VMs to ensure the optimal utilization of the system. Finally, for

MSLBL_MW, we assumed that every VM can contain software configurations for every workflow application type and can be shared between any users in WaaS platforms.

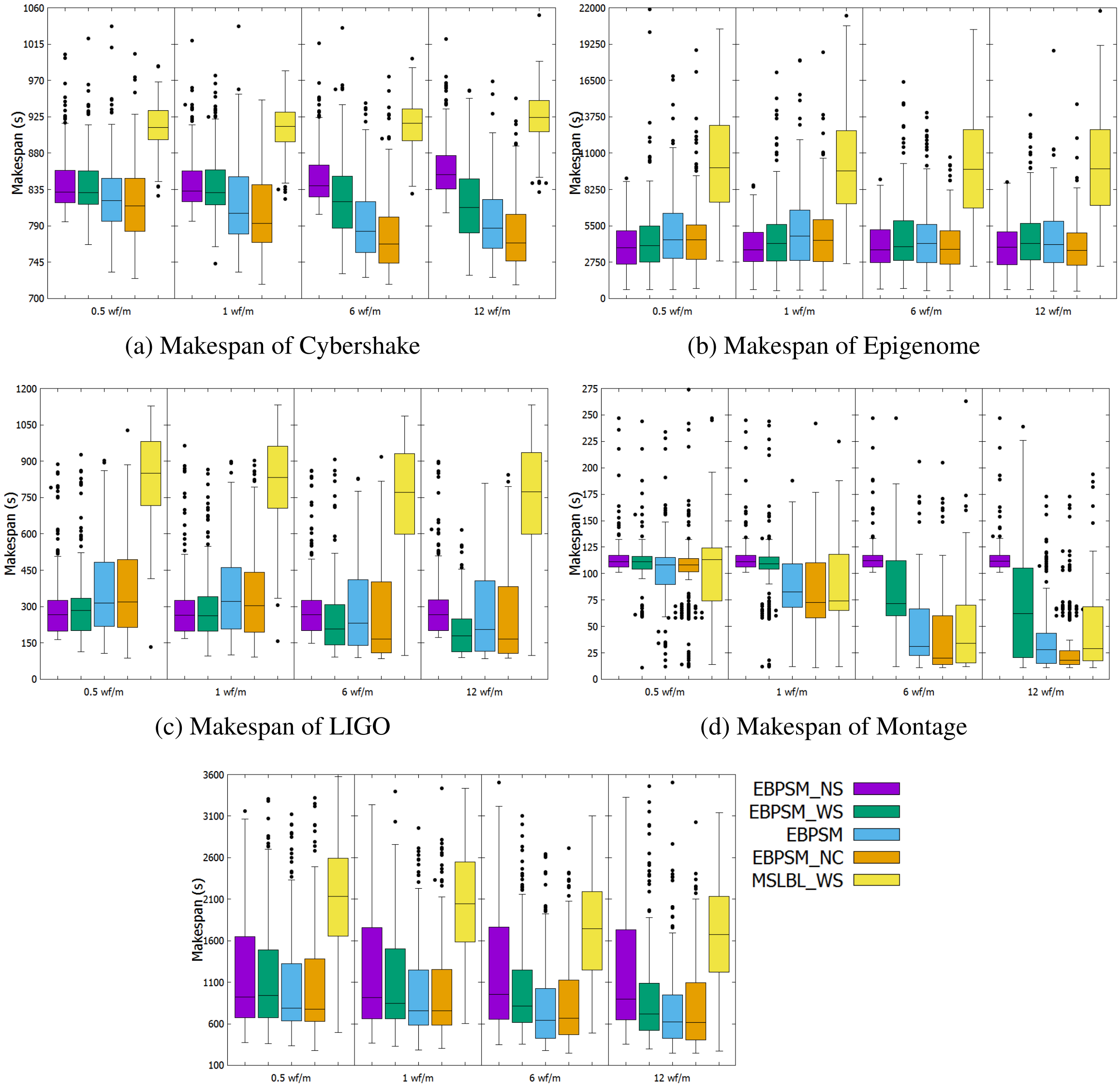
To demonstrate the benefits of our policy, we implemented three additional versions of EBPSM, which are EBPSM_NS, EBPSM_WS, and EBPSM_NC. EBPSM_NS does not share the VMs between different users; it is a version of EBPSM that executes each workflow submitted into the WaaS platform in dedicated service, as discussed in Chapter 1 shown in Fig. 1.3a. EBPSM_WS tailors the software configuration of workflow applications in a VM image instead of containers. Therefore, the algorithm allows only tasks from the same workflow application type (e.g., SIPHT with 50 tasks and 100 tasks) that can share the provisioned VMs during the execution.

Meanwhile, EBPSM_NC ignores the use of containers to store the configuration template and naively assumes that each VM can be shared between many users with different requirements. This version was a direct comparable case for MSLBL_MW. Finally, the $threshold_{idle}$ for EBPSM_WS, EBPSM, and EBPSM_NC was set to 5 seconds. It means the vm_{idle} is not immediately terminated whenever it goes idle to accommodate the further utilization of the cached data within the VM.

4.5.1 To Share or Not To Share

The purpose of this experiment is to evaluate the effectiveness of our proposed resource-sharing policy regarding its capability to minimize the workflows' makespan while meeting the soft limit budget. In this scenario, we evaluated EBPSM and its variants against MSBLBL_MW under four workloads with a different arrival rate of 0.5, 1, 6, and 12 workflows per minute. Each workload consists of 1000 workflows with approximately 170 thousand tasks of various sizes (e.g., small, medium, large) and different workflow types generated randomly based on a uniform distribution. The arrival rate for these four workloads represents the density of workflows' arrival in the WaaS platform. The arrival of 0.5 workflows per minute represents a less occupied platform, while the arrival of 12 workflows per minute models a busier system in handling the workflows.

Fig. 4.1a, 4.1b, 4.1c, 4.1d, and 4.1e depict the makespan achieved for Cybershake, Epigenome, LIGO, Montage, and SIPHT workflows respectively. EBPSM_NS that represents the traditional non-shared cloud resources paradigm shows almost no difference in the algorithm's performance across different arrival rates. This version of the algorithm serves each of the workflows in dedicated and isolated resources. Therefore, it can maintain a similar performance for all four work-



(a) Makespan of Cybershake

(b) Makespan of Epigenome

(c) Makespan of LIGO

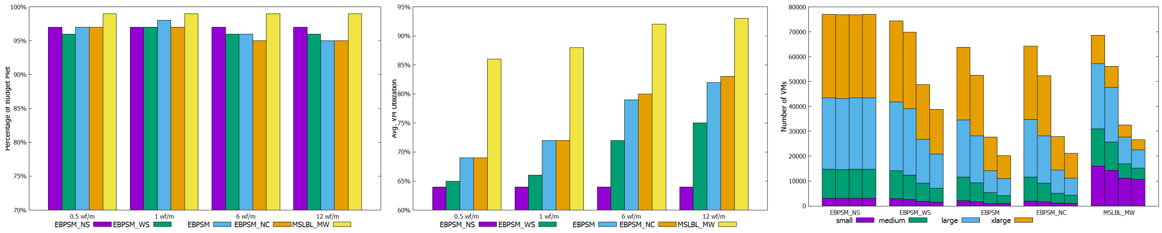
(d) Makespan of Montage

(e) Makespan of SIPHT

Fig. 4.1: Makespan of workflows on various workloads with different arrival rate

loads. However, on the other hand, EBPSM_NS shows the lowest percentage of average VM utilization due to this non-sharing policy, as seen in Fig. 4.2b.

In contrast to EBPSM_NS, the other three versions of EBPSM exhibit a performance improvement along with the increasing density of the workloads. This further makespan minimization is the result of (i) the elimination of data transfer overhead between tasks' execution and (ii) the utilization of inevitable scheduling gaps between tasks' execution. In the case of data transfer elimination, we can observe that the improvement is relatively not significant for Epigenome



(a) Percentage of budget met

(b) Avg. VM utilization

(c) Number of VMs

Fig. 4.2: Percentage of budget met and VM usage on various workloads with different arrival rate

workflows, where the CPU processing takes the most significant portion of the execution time instead of I/O and data movement. On the other hand, the most significant improvement can be observed from the data-intensive workflows such as Montage and Cybershake applications.

Furthermore, the superiority of EBPSM and EBPSM_NC over EBPSM_WS both in makespan and average VM utilization shows a valid argument for the schedule gaps utilization case. From the result, we can conclude that the idle gaps utilization between users from different workflows can further minimize the makespan. The naive assumption that every VM can be shared between any users in the platform explains the lower makespan and the higher utilization of EBPSM_NC compared to EBPSM. Container usage generates additional delays that affect EBPSM performance. However, the difference between them is marginal, and EBPSM still exhibits a good result.

We observe that in four out of five cases, all versions of EBPSM overthrow MSLBL_MW regarding the makespan achievement. This result comes from the different strategies of both algorithms in distributing the budget and avoiding the constraint's violation, which implies the type of VMs they provisioned. EBPSM prioritizes the budget allocation to the earlier tasks and leases the fastest VM type as much as possible. This approach is based on the idea that the following children's tasks can utilize already provisioned VMs while maintaining the capability of meeting the budget by updating the allocation based on the actual tasks' execution.

In Fig. 4.2a, we can see that all algorithms can achieve at least 95% of the budget meeting for all cases. The margin between MSLBL_MW and the four versions of EBPSM was never wider than 4%. Furthermore, MSLBL_MW is superior to EBPSM regarding the average VM utilization, as seen in Fig. 4.2b. This result is caused by the difference in the VM deprovisioning policy. MSLBL_MW eliminates any VMs as soon as they become idle. At the same time, EBPSM delays

Table. 4.3: Cost/budget ratio for EBPSM budget violated cases

Percentile	**0.5 wf/m**	**1 wf/m**	**6 wf/m**	**12 wf/m**
10th	1.005	1.004	1.017	1.005
30th	1.017	1.018	1.032	1.023
50th	1.026	1.030	1.051	1.052
70th	1.046	1.053	1.065	1.069
90th	1.072	1.083	1.121	1.107

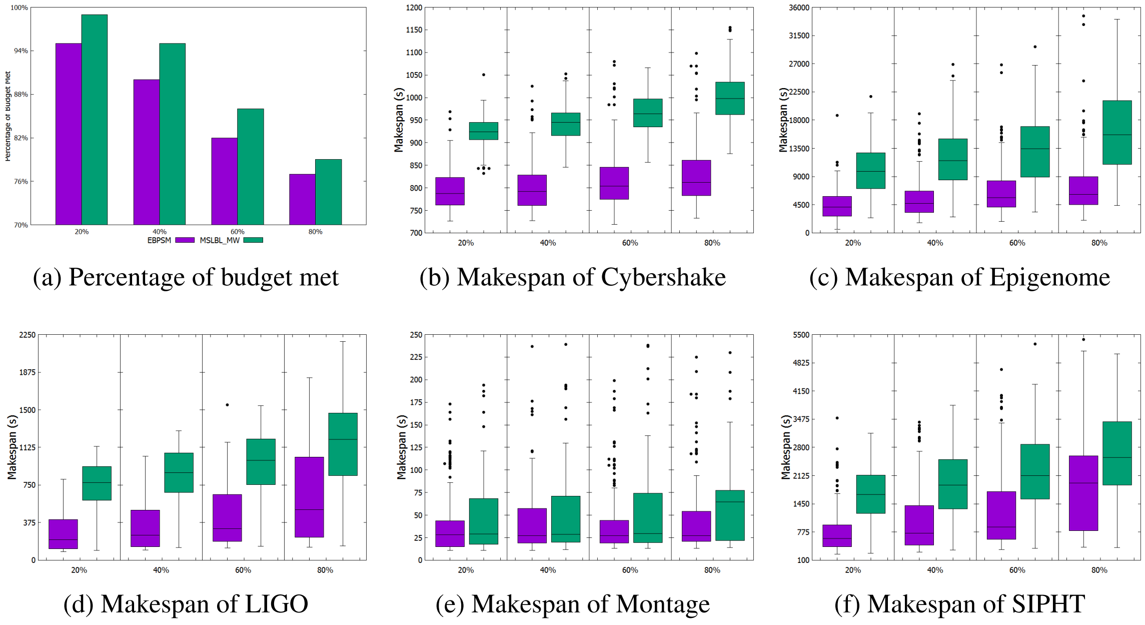
the elimination in the hope of further utilization and cached data for the following tasks on that particular idle VM. In this case, the configurable settings of $threshold_{idle}$ value may affect the VM utilization. However, from these two approaches, a significant margin of makespan between MSLBL_MW and EBPSM can be observed in most cases.

On the other hand, MSLBL_MW allocates the budget based on the budget level factor, which creates a safety net by provisioning the VM that costs somewhere between the minimum and maximum execution cost of a particular task. In this way, MSLBL_MW reduces the possibility of budget violations at the budget distribution phase. These two different approaches result in the different number of VM types used during the execution, as can be seen from Fig. 4.2c. MSLBL_MW leases a lower number of faster VM types compared to EBPSM for all cases. The only case where the performance of MSLBL_MW is relatively equal to EBPSM is in the Montage workflow, where the tasks are relatively short in CPU processing time. In contrast, the significant portion of their execution time takes place in the data movement. In this Montage case, the decision to lease which kind of VM type does not significantly affect the total makespan.

We captured the cases where EBPSM failed to meet the budget and presented the result in Table 4.3 to understand the EBPSM performance. From the table, we can see that the cost/budget ratio for 90% of the EBPSM budget violation cases is lower than 1.12. This ratio means that the additional cost from these violations is never higher than 12%. This percentage is relatively small and may be caused by an extreme case of CPU and network performance degradation. In addition, to eliminate the negative impact of the uncertainties in such a dynamic environment is not possible.

4.5.2 Performance Degradation Sensitivity

Adapting to performance variability is an essential feature for schedulers in multi-tenant dynamic environments. This ability ensures the platform can quickly recover from unexpected events that may occur at any given time, hence preventing a snowball effect that negatively impacts subse-



(a) Percentage of budget met

(b) Makespan of Cybershake

(c) Makespan of Epigenome

(d) Makespan of LIGO

(e) Makespan of Montage

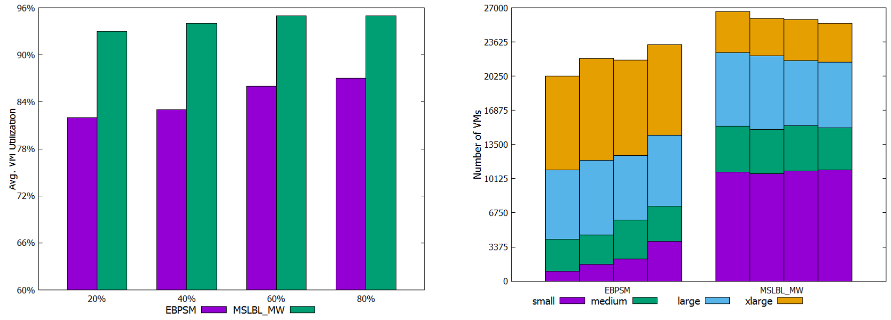
(f) Makespan of SIPHT

Fig. 4.3: Percentage of budget met and makespan of workflows on various CPU performance degradation

quent executions. In this experiment, we evaluate the sensitivity of EBPSM and MSLBL_MW–on the default environment–to CPU performance degradation by analyzing the percentage of budgets met, makespan, average VM utilization, and the number of VMs used on four different scenarios. We model CPU performance degradation using a normal distribution with a 1% variance and different average and maximum values. The average value is defined as half of the maximum of the CPU performance degradation, which ranges from 20% to 80%.

All algorithms are significantly affected by CPU performance degradation as their percentage of budget met decreases along with the increased maximum degradation value. However, MSLBL_MW suffers the most as its performance margin with EBPSM is getting smaller, as seen in Fig. 4.3a. This suffering also can be observed in Fig. 4.3b, 4.3c, 4.3d, 4.3e, 4.3f. The increasing makespan as a response to the performance degradation for EBPSM was relatively lower than its effect on MSLBL_MW. EBPSM can perform better than MSLBL_MW because of its capability to adapt the changes by evaluating a particular task's execution right after it was finished.

Meanwhile, MSLBL_MW only relies on the spare budget from its safety net of budget allocation that limits the number of faster VM types at the budget distribution phase. When the maximum degradation value increases, there is no extra budget left from this safety net. Hence,



(a) Avg. VM Utilization

(b) Number of VMs

Fig. 4.4: VM usage on various CPU performance degradation

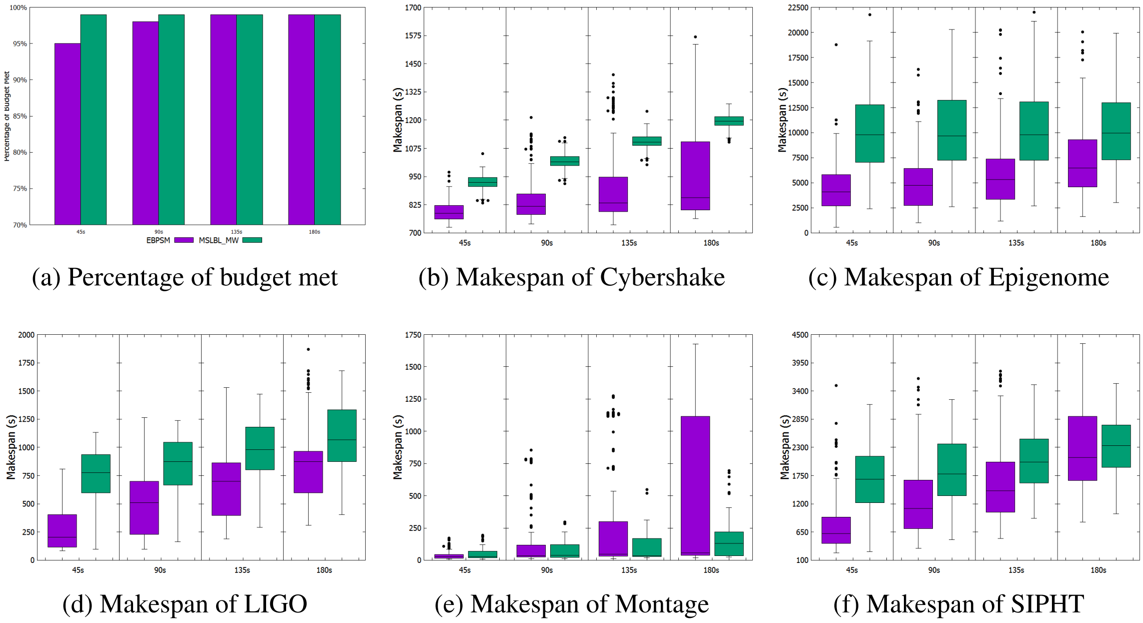
the ability of MSLBL_MW to meet the budget drops faster than EBPSM. This reason is also in line with the average VM utilization results where MSLBL_MW cannot increase VMs utilization as it reaches the top limit of its capabilities, as seen in Fig. 4.4a.

Another perspective can be observed from the number of VMs used by each algorithm, as depicted in Fig. 4.4b. EBPSM recovers from the CPU performance degradation by continually increasing the number of slower VM types while maintaining the faster VM types as much as possible. This adaption is being made through the budget update process after a task finished being executed. Therefore, the earlier tasks are still able to be scheduled onto faster VMs.

In contrast, the children's tasks in the tail of the workflow are inevitably allocated to the slower ones due to the budget left from the recovering process. Different behaviour is observed from MSLBL_MW usage of VMs. We cannot see any significant effort to recover from the number of VM type used. We argue that MSLBL_MW's way of dealing with CPU performance degradation is by highly relying on the safety net of budget allocation from the budget distribution phase.

4.5.3 VM Provisioning Delay Sensitivity

A large volume of workflows results in a vast number of tasks in the scheduling queue to be processed at a given time. Regardless of the effort put into minimizing the number of VMs used by sharing and reusing already provisioned VMs, provisioning a large number of VMs is inevitable in the WaaS platform. This provisioning becomes a problem if the scheduler is not designed to handle the uncertainties from delays in acquiring the VMs. In this experiment, we study the sensitivity of EBPSM and MSLBL_MW algorithms–on the default environment–towards VM provisioning delay by analyzing the budget met percentage, makespan, average VM utilization, and the number



(a) Percentage of budget met (b) Makespan of Cybershake (c) Makespan of Epigenome

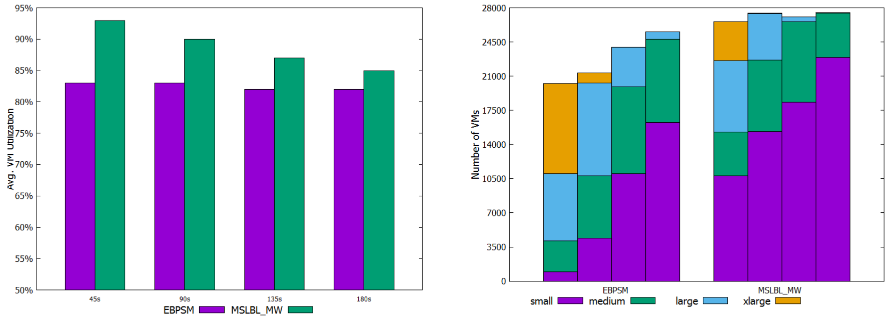
(d) Makespan of LIGO (e) Makespan of Montage (f) Makespan of SIPHT

Fig. 4.5: Percentage of budget met and makespan of workflows on various VM provisioning delay

of VMs used on four delay scenarios. The delays range from 45 to 180 seconds.

An interesting point can be observed in Fig. 4.5a that shows the budget meeting percentage in the experiments. All of the algorithms perform well in facing VM provisioning delays. However, this well-performed result comes with a trade-off of the workflows' makespan, as seen in Fig. 4.5b, 4.5c, 4.5d 4.5e, and 4.5f, especially for the workflow applications that highly depend on CPU processing. We can observe that the spectrum for Montage and Cybershake makespan in EBPSM for 180 seconds delays is quite wide, although its overall performance is still superior to MSLBL_MW. In contrast to this situation, the makespan spectrum for MSLBL_MW for those two workflows is very narrow.

One of the reasons that may precipitate this broad spectrum of makespan for EBPSM is the variation in the number of shared VMs and the types of VM used during the execution. Notably, the disparity of the makespan is broader for the higher VM provisioning delay scenarios. On the other hand, MSLBL_MW adopts a simple policy in sharing and reusing already provisioned VMs. This algorithm did not take the possible data sharing into consideration, whether a particular VM contains a cached data for a future scheduled task–it merely scheduled the task to any idle VM available. In this way, the MSLBL_MW algorithm can reduce the variation of the type of VM used during the workflows' execution.



(a) Avg. VM Utilization

(b) Number of VMs

Fig. 4.6: VM usage on various CPU provisioning delay

When EBPSM leases faster VMs for the earlier tasks, the VM provisioning delays increase the actual cost for those VMs. Therefore, the algorithm cannot afford those already provisioned VMs if the budget left for the following tasks is not enough to provision the same VM type. In this case, EBPSM must provision a new VM with a slower type. Hence, a workflow with a tight budget will suffer unexpected longer makespan. This condition happens as the algorithm tries to allocate a faster VM type for the earlier tasks, but it turns out that the budget left is not sufficient. Therefore, to avoid the budget violation, the algorithm automatically recovers during the budget redistribution by provisioning a new slower VM type for that particular task. Fig. 4.6b confirms this situation where EBPSM used a wider variety of VM types compared to MSLBL_MW. However, in general, the delays do not directly affect the average VM utilization, since the VM initiating process is not counted through the total utilization. However, still, these delays are being charged. Therefore, that is why the delay profoundly affects the overall budget. Finally, this analysis does not hinder the fact that EBPSM, in general, is still superior to MSLBL_MW in terms of makespan.

4.5.4 Container Initiating Delay Sensitivity

Various applications consist of a different number of tasks and software dependencies that may impact the size of the container images. Therefore, in this experiment, we study the sensitivity of the EBPSM algorithm–on the default environment–toward container initiating delay by analyzing the percentage of budget met and makespan with five different container initiating delays. The delays range from 10 to 50 seconds.

In this experiment, we find that the budget correction of EBPSM can maintain budget-met compliance of 94-95% in all scenarios. Therefore, we do not plot the budget-met percentage

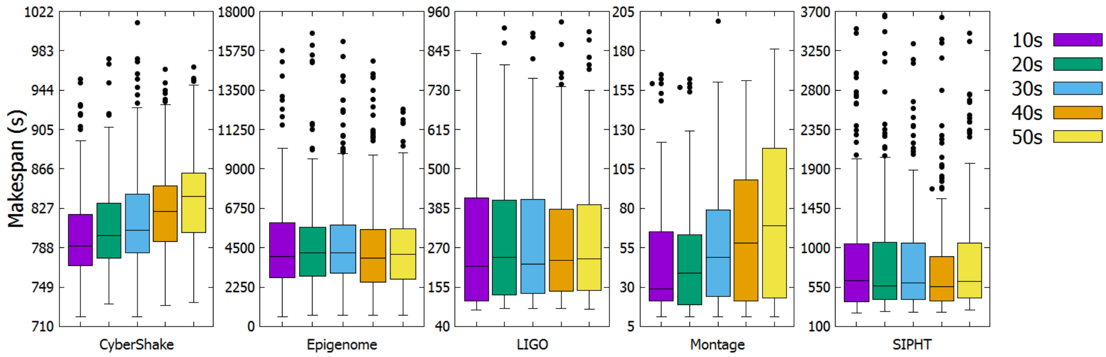


Fig. 4.7: EBPSM performance on various container initiating delay

and present the impact to the makespan instead. From Fig. 4.7, we can see that the container initiating delay affects the workflows with a high number of tasks and I/O requirements. In this case, Cybershake and Montage makespan increases along with the delays, while the results for the other workflows are relatively indistinguishable.

From these two experiments, we can observe that the container initiating delays may have a higher performance impact to the workflows with a high number of parallel tasks. This condition may be caused by the necessity of deploying a high number of container images to cater to the parallel execution. In this case, a decision to store particular container images of these workflows must be carefully considered. Therefore, a more sophisticated strategy of maintaining container image locality is a crucial aspect to be explored.

4.6 Summary

The growing popularity of scientific workflows deployment in clouds drives the research on multi-tenant platforms that provides utility-like services for executing workflows. As well as any other multi-tenant platforms, this WaaS platform faces several challenges that may impede the system's effectiveness and efficiency. These challenges include the handling of a continuously arriving workload of workflows, the potential of system inefficiency from inevitable idle time slots from workflows' tasks dependency execution, and the uncertainties from computational resources performances that may impose significant overhead delays.

WaaS platforms extend the capabilities of a traditional Workflow Management System (MWS) to provide a more comprehensive service for more significant users. In a conventional WMS, a single workflow job is processed in dedicated service to ensure its Quality of Service (QoS) requirements. However, this approach may not be able to cope with the WaaS platforms. A significant deficiency may arise from its conventional way of tailoring workflows' software configurations into a VM image, inevitable intra-dependent workflows' tasks schedule gaps, and possible overhead delays from workflow pre-processing and data movement handling.

To achieve more efficient WaaS platforms, we propose a resource-sharing policy that utilizes container technology to wrap software configurations required by particular workflows. This strategy enables (i) the idle slots VMs sharing between users and (ii) gaining further makespan minimization by sharing the datasets and container images cached within VMs local storage. We implement this policy on EBPSM, a dynamic heuristic scheduling algorithm designed for multiple workflows scheduling in the WaaS platform. Our experiments show that the sharing scenarios overthrow a traditional dedicated approach in handling workflows' jobs. Furthermore, our proposed algorithm can surpass a modified state-of-the-art budget-constrained dynamic scheduling algorithm in terms of minimizing the makespan and meeting the budget.

There are several aspects of our experiments that need to be further investigated. The budget distribution phase plays a vital role in budget-constrained scheduling. The decision to either allocate more budget for the earlier tasks so they can lease faster computational resources or maintain a safety net allocation to ensure the budget compliance must be carefully taken into account. A trade-off between having a faster execution time and meeting the allocated budget is always inevitable. In this way, defining the nature of execution, including strictness of the budget constraints, can help to design an appropriate configuration between two approaches.

Furthermore, the resource provisioning strategy must consider the *quid pro quo* between having a higher system utilization (i.e., lower idle VM times) and an optimal data sharing and movement which utilizes the VM local storage. We observe that delaying a particular VM termination may improve the performance when the cached data stored within the VM is intelligently considered. In this chapter, we do not consider task failure either caused by the software or the infrastructure (i.e., container, VMs). Incorporating a fault-tolerant strategy into both EPSM and EBPSM algorithms is necessary to address the task failure that is highly likely to create an impact on the WaaS platforms' performance.

Finally, an investigation on how multiple container instances can be run and scheduled on top of a single VM is another to-do list. The delay in initiating a container image has reduced our algorithm's performance. There must be a way to counterbalance this issue by exploiting the swarming nature of containers to gain benefits from this inevitable state to enhance the efficiency further. Up to this chapter, we evaluate our proposal using a simulation approach, which assumed that the estimation of tasks' runtime is always available before the scheduling. In the next chapter, we propose an online incremental machine learning approach to handle the task runtime prediction in the WaaS platform.

This page intentionally left blank.

Chapter 5

An Online Incremental Learning Approach for Task Runtime Estimation

In this chapter, we propose an online incremental learning approach to predict the tasks' runtime of scientific workflows in cloud computing environments. To improve the performance of the predictions, we harness fine-grained resources monitoring data in the form of time-series records of CPU utilization, memory usage, and I/O activities that are reflecting the unique characteristics of a task's execution. We compare our solution to a state-of-the-art approach that exploits the resources monitoring data based on the regression machine learning technique. From our experiments, the proposed strategy improves the performance, in terms of the error, up to 29.89%, compared to the state-of-the-art solutions.

5.1 Introduction

Recent studies show a plethora of algorithms were designed to schedule scientific workflows in cloud computing environments [122]. The majority of the solutions are based on heuristic and metaheuristic approaches that attempt to find a near-optimal solution to this NP-hard problem. These optimization techniques in scheduling rely on the estimation of task runtime and resource performance to make scheduling decisions. This estimate is vital, especially in a cost-centric, dynamic environment like clouds. An inaccurate estimate of a task's runtime in scientific workflows has a snowball effect that may eventually lead to all of the successors of the task taking a longer time than expected to complete. In the end, this will have a negative impact on the total workflow execution time (i.e., makespan) and inflict an additional cost for leasing the cloud resources.

This chapter is derived from: **Muhammad H. Hilman**, Maria A. Rodriguez, and Rajkumar Buyya. 'Task Runtime Prediction in Scientific Workflows Using an Online Incremental Learning Approach.' *In Proceedings of the 11th ACM/IEEE International Conference on Utility and Cloud Computing (UCC)*, Pages 93-102, 2018.

With the emergence of the WaaS platforms that deal with a significant amount of data, having a module within the system that can predict the task's runtime in an efficient and low-cost fashion is an ultimate requirement. In this case, the workload of workflows must be handled as soon as they arrive due to the quality of service (QoS) constraints defined by the users. Hence, these platforms need to be capable of processing requests in a near real-time fashion. The runtime prediction of tasks must be achieved in a fast and reliable way due to the nature of the environment. Moreover, WaaS platforms make use of the distributed resources provided by the Infrastructure-as-a-Service (IaaS) provider. Therefore, the prediction method should be able to adapt to a variety of IaaS cloud infrastructures seamlessly.

Predicting task runtime in clouds is non-trivial, mainly due to the problem in which cloud resources are subject to performance variability [37]. This variability occurs due to several factors–including virtualization overhead, multi-tenancy, geographical distribution, and temporal aspects [4]–that affect not only computational performance but also the communication network used to transfer the input/output data [119]. In this area, most of the existing approaches are based on the profiling of tasks using basic statistical description (e.g., mean, standard deviation) to summarize the existing historical data of scientific workflow executions to characterize the tasks, which then is exploited to build a performance model to predict the task runtime. Another approach uses a profiling mechanism that executes a task in a particular type of resource and utilizes the measurement as an estimate. These methods are impractical to adopt in cloud computing environments. Relying only on profiling based on the statistical description does not capture sudden changes in the cloud's performance. For example, it is not uncommon for a task to have a longer execution time during a specific time in cloud instances (i.e., peak hours). Hence, averaging the runtime data without considering the temporal factors will only lead to inaccurate predictions. Meanwhile, profiling tasks by executing them in the desired type of resources will lead to an increase in the total execution cost because the profiler requires an extra budget to estimate the runtime.

On the other hand, machine learning approaches can be considered as a state-of-the-art solution for prediction and classification problems. Machine learning approaches learn the relation between a set of input and its related output through intensive observation from characteristics of the data, usually referred to as features. To capture several aspects that affect the cloud's performance variation, machine learning may provide a better solution by considering temporal changes and other various factors in the task's performance as features.

In the case of predictions, the conventional machine learning approaches are based on a regression function that estimates the runtime of a task from a set of features. Evaluating these techniques to predict the task runtime in WaaS platforms is out of our scope. We are interested in exploring various ways of determining the features on which the regression functions depend on. Typical variables that are being used as features to predict the task runtime are based on the workflow application attributes (e.g., input data, parameters) and the specific hardware details (e.g., CPU capacity, memory size) in which the workflows are deployed. This information is relatively easy to extract, and their values are available before the runtime. However, with the rising trend of cloud computing to deploy the scientific workflows, some of these variables that are related to the specific hardware details may become inaccurate to represent the computational capacity due to the performance variability of cloud instances.

Moreover, we found that, as a part of the anomaly detection in executing scientific workflows, some WMS is equipped with the capability to monitor the runtime of tasks by collecting their resource consumptions in a time-series manner. This approach is a more advanced approach than a typical resource consumption monitoring method that stores only the single value of the total resource usage of a task's execution. We argue that time-series data of a task's resource consumption may represent better information of a task's execution to be used as features.

Based on these requirements, we propose an online incremental learning approach for task runtime prediction of scientific workflows in cloud computing environments. The online approach can learn as data becomes available through streaming and considered as fast as the model only sees and processes a data point once when the task finishes. The incremental approach enables the model to capture the changes such as peak hours in clouds and is capable of adapting to the heterogeneity of different IaaS cloud providers. We also propose to utilize resource monitoring data such as memory consumption and CPU utilization that is collected continuously based on a configurable time interval of time-series records.

This chapter is organized as follows. Section 5.2 reviews works that are related to our study. Section 5.3 describes the problem definition. Meanwhile, Section 5.4 explains online incremental learning and Section 5.5 describes the proposed solution. Performance evaluation is presented in Section 5.6 and Section 5.7 analyzes the results. Finally, Section 5.8 summarizes the findings.

5.2 Related Work

The profiling of the task's performance in scientific workflows has been extensively studied to create a model that can be used to estimate runtime. The work is useful for improving the scheduling of workflows as the estimation accuracy affects the precision of scheduling algorithms' performance. A study by Juve et al. [27] discussed the characterization and profiling of workflows based on the system usage and requirements that can be used for generating a model for estimating the task runtime based on a basic statistical description. Our work differs in that we use machine learning to predict task runtime instead of the statistical description to summarize the data.

Another work used evolutionary programming in searching the workflow execution similarities to create a template based on the workflow structure, application type, execution environment, resource state, resource sharing policy, and network [123]. The template that refers to a set of selected attributes of scientific workflow execution is later used to generate a prediction model for task runtime in a grid computing environment. The use of evolutionary programming is known for its compute-intensiveness as the search space increases. It differs from our work, which is based on an online approach to achieve fast predictions.

A runtime estimation framework built for ALICE (**A** **L**arge **I**on **C**ollider **E**xperiment) profiled a sample of tasks by executing them before the real workflow execution to predict their runtime [124]. The framework captures the features of sample task execution records and uses them as input for the model. This approach is suitable for massive established computational infrastructures, but may not be appropriate for cloud computing environments. Our work considered clouds; therefore, we avoid additional costs as much as possible by doing extra execution.

The works that consider machine learning approaches are dominating the state-of-the-art task runtime prediction. Regardless of the type of machine learning techniques that are being used, the proposal by Da Silva et al. [35] exploited the workflow application attributes–such as input data, parameters, and workflow structure–as features to build the prediction models. These attributes uniquely adhere to the tasks and are available before the execution. However, in WaaS platforms where the resources are leased from third-party IaaS providers, the hardware heterogeneity may result in different performance even for the same task of the workflow.

In this case, the other works combine the workflow application attributes with specific hardware details where the workflows are deployed, as features. Matsunaga and Fortes [125] used application attributes (e.g., input data, application parameters) in combination with the system attributes (e.g., CPU microarchitecture, memory, storage) to build the prediction model for resource consumption and task runtime. Another work by Monge et al. [126] exploited the task's input data and historical provenance data from several systems to predict task runtime in the gene-expression analysis workflow. The combination of application and hardware attributes provides better profiling of the task's execution that arguably results in the improvement of task runtime prediction.

However, only using features for which the values are available before runtime–such as application attributes and hardware details–may not be sufficient to profile the task's execution time in cloud environments. Therefore, further works in this area consider the specific execution environment (e.g., cloud sites, submission time) and the resource consumption status (e.g., CPU utilization, memory, bandwidth) as features. Some of these may be available before runtime (e.g., execution environment), but most of them (e.g., resource consumption status) can only be accessed after the task's execution. Hence, the latest variables are mostly known as runtime features as their collection occurs during the task's execution. This runtime features plausibly provide better profiling of the task's execution in cloud environments. The works that exploit this approach, such as Seneviratne and Levy [127], used linear regression to estimate this runtime features–such as CPU and disk load–before using them to predict the task runtime. Meanwhile, Pham et al. [36] proposed a similar approach, called two-stage prediction, to estimate the resources consumption (e.g., CPU utilization, memory, storage, bandwidth, I/O) of a task's execution in particular cloud instances before exploiting them for task runtime prediction.

Nonetheless, these related works are based on batch offline machine learning approaches. The batch offline approach poses an inevitable limitation in WaaS platforms. This limitation is related to the streaming nature of workloads in WaaS platforms that need to be processed as soon as they arrive in a near real-time fashion. Our work differs in that we use an online incremental approach and exploit the time-series resource monitoring data to predict the task runtime.

5.3 Problem Definition

In this chapter, we focused on the task runtime estimator. We assumed running tasks are continuously monitored to measure their resource consumption in a specific computational resource. Different metrics capture the usage of different resources, such as CPU, memory, and I/O. These

Table. 5.1: Description of runtime resource consumption metrics

Resource	**Metric**	**Description**
CPU	procs	Number of process
	stime	Time spent in user mode
	threads	Number of threads
	utime	Time spent in kernel mode
Memory	vmRSS	Resident set size
	vmSize	Virtual memory usage
I/O	iowait	Time spent waiting on I/O
	rchar	Number of bytes read using any read-like syscall
	read_bytes	Number of bytes read from disk
	syscr	Number of read-like syscall invocations
	syscw	Number of write-like syscall invocations
	wchar	Number of bytes written using any write-like syscall
	write_bytes	Number of bytes written to disk

are described in Table 5.1. As a result, the data collected for each task and each metric correspond to a series of tuples consisting of a timestamp t and a value v ($< t, v >$), where the value corresponds to a specific resource consumption measurement. The measurement's frequency is configurable by a time interval τ. Smaller τ values translate into more frequent measurements, while larger values reduce the frequency and result in less monitoring data. These time-series records are stored in a monitoring database, which is later used by the task runtime estimator.

We also assumed some features describing a given task and its execution environment are available. In particular, the task's profile, virtual machine configuration used for the task's execution, and the task's submission time. These are shown in Table 5.2 and are referred to from now on as pre-runtime features. The problem becomes then on efficiently utilizing these pre-runtime data in conjunction with the resource monitoring time-series data to accurately estimate the runtime of a task in an online incremental manner, that is, as it arrives for execution.

5.4 Online Incremental Machine Learning

In general, machine learning methods are employed to learn some insights from patterns in available data and to predict future events. Classical batch offline learning, which learns from an already collected and accessible dataset, is not suitable for processing a rapid volume of data in a short time. A reason for this is the fact that these methods usually require the entire dataset to be

Table. 5.2: Description of pre-runtime configuration

	Name	Description
Task	name	Name of the task
	id	ID for a particular type of task
	input	Input name for a task
VM Type	memory	Memory capacity
	storage	Storage capacity
	vcpu	Number of virtual processor
Submission Time	day	submission day
	hour	submission hour

loaded into memory. Furthermore, batch offline methods do not continuously integrate additional information as the model incrementally learns from new data, but instead reconstruct the entire model from scratch. This approach is not only time consuming and compute-intensive, but also may not be able to capture the temporal dynamic changes in the data statistics. As a result, batch offline learning methods are not appropriate for dynamic environments involving a significant amount of data in a streaming way, such as WaaS platforms.

Instead, online incremental learning has gained significant attention with the rise of big data and the internet of things (IoT) as it deals with a vast amount of data that did not fit into memory and may come in a streaming fashion. As a result, we propose the use of two algorithms implemented using online incremental learning approaches to estimate the runtime of tasks in a near real-time way, Recurrent Neural Network (RNN) and K-Nearest Neighbors (KNN). Online incremental learning methods fit naturally into WaaS platforms since they incrementally incorporate new insights from new data points into the model and traditionally aim to use minimal processing resources as the algorithms read the new data once available. The other advantage worth noting is that incremental learning enables the model to adapt to different underlying infrastructures. Hence, it allows the creation of models that are agnostic to platforms.

5.4.1 Recurrent Neural Networks

A particular type of RNN called Long Short-Term Memory networks (LSTMs) is capable of remembering information for an extended time [128]. Instead of having a simple layer as in regular RNNs, an LSTM network has four unique plus one hidden layers in repeating modules that enable them to learn the context and decide whether the information has to be remembered or forgotten.

These layers are a memory unit layer c, three types of gate layers- the input gate i, the forget gate f, and the output gate o- plus a hidden state h.

For each time step t, LSTM receives a set of values x_t corresponding to the different features of the data, and the previously hidden state h_{t-1} that contains the context from previous information as input. Then, LSTM computes the output of the gates based on the activation function which includes the weights and biases of each gates. Finally, this process can be repeated and configured for it to produce an output sequence $\{o_t, o_{t+1}, o_{t+2}, o_{t+3}, . . ., o_{t+n}\}$ as the prediction result.

Based on these capabilities, LSTM is suitable and shows promising results for time-series analysis [129]. Moreover, it supports online learning since its implementation in Keras[1] provides *batch size* variable that limits the number of data to be trained. A *batch size* value of 1 is used for an online learning approach. Keras also accommodates incremental learning as it incorporates the ability to update the LSTM model whenever new information is obtained continuously.

5.4.2 K-Nearest Neighbor

K-Nearest Neighbor (KNN) is a machine learning algorithm that generates classification/prediction by comparing new problem instances with instances seen in training. KNN computes distances or similarities of new instances to the training instances when predicting a value/class for them. Given a data point x, the algorithm computes the distance between that data and the others in the training set. Then, it picks the nearest K training data and determines the prediction result by averaging the output values of these K points.

KNN is widely used for prediction in many areas from signal processing [130] to time-series analysis [131] that resembles sequential problems. One of the implementations of KNN is the IBk algorithm [132] that is included in the WEKA data mining toolkit [133]. It incorporates the capability to learn the data incrementally. While the lazy behaviour of KNN is compute-intensive and may slow down the performance as the training set increases, IBk incorporates a *window size* variable that enables the algorithm to maintain a number of records from the training set. This capability achieves the trade-off between learning accuracy and speed that is determined by the size of the window by dropping the older data as new data is added to the set. This essential function becomes an advantage for the algorithm to handle the changes in the statistics of the data.

[1] https://keras.io/

Algorithm 6 Task runtime prediction

```
Input: a task of the workflow $t_i$
Input: a virtual machine type $v_i$
Input: submission time $s_i$
Output: runtime prediction $\alpha$ for $t_i$ on $v_i$ at $s_i$
while incoming task $t$ in WaaS do
    Phase 1:
    $\sigma_i \leftarrow$ extract pre-runtime features for $t_i$ on $v_i$ at $s_i$
    Phase 2:
    for selected runtime features $R$ of task $t_i$ do
        $\{r_{j_1}, r_{j_2}, .., r_{j_n}\} \leftarrow$ predict resource consumption $r_j$ of $t_i$ using $\sigma_i$
        $\varsigma_j \leftarrow$ extract feature of time-series $\{r_{j_1}, r_{j_2}, .., r_{j_n}\}$ using Eq. 5.1
    Phase 3:
    $\alpha \leftarrow$ predict runtime of $t_i$ using $\sigma_i$ and a set of features $\{\varsigma_1, \varsigma_2, .., \varsigma_n\}$ from $R$
```

5.5 Task Runtime Prediction Using Time-Series Monitoring Data

In this chapter, we aim to predict the runtime of the task in a WaaS platform. Given a set of pre-runtime features as listed in Table 5.2, we built a model using an online incremental approach that can provide an estimate of the time needed to complete a task in a specific virtual machine. In particular, we implemented a task runtime estimator module that can be easily plugged in into a WaaS platform and the only requirements being (i) access to the pre-runtime features of a task and (ii) a resource consumption monitoring system that records data in a time-series database. We made use of these data to build the model as a task finishes its execution incrementally. Specifically, when a task is fed into the WaaS scheduler, the algorithm extracts its pre-runtime features and predicts its resource consumption estimation for each metric using LSTM. Then, each resource consumption of a task (i.e., first phase prediction result) is processed to get a representative and distinctive value from the time-series. This process is called feature extraction. Afterward, this value from the feature extraction, along with the pre-runtime features are fed to IBk to predict the task's runtime. This process is outlined in Algorithm 6. From now on, we refer to our proposed approach as the time-series scenario.

We proposed a framework in which multiple prediction models, one for each task in the workflow, are maintained, rather than having a single prediction model for all tasks submitted into the system. We argue that this approach has three main benefits (i) a single prediction model contains information that may act as noise for different tasks, (ii) the size of a single model will grow as the number of tasks increases; this may not be scalable to the size of memory, (iii) multiple models can be maintained by temporarily saving unused models into disk and being loaded whenever the corresponding task needs to be processed. Furthermore, multiple models allow the system to optimize predictions as each model can be fine-tuned to a specific setting (e.g., feature selection).

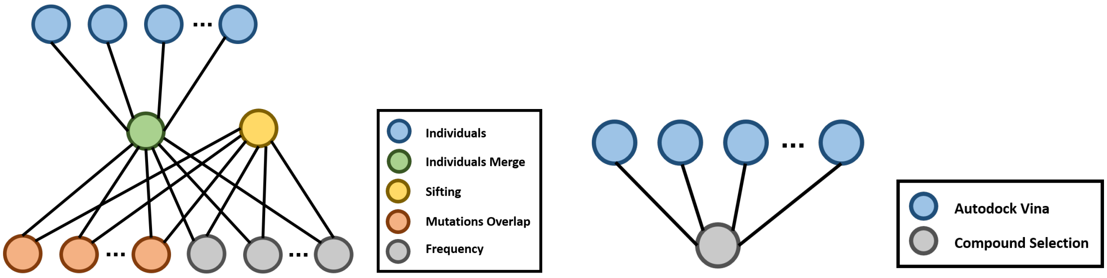
To execute the workflows and collect the monitoring data, we used the Pegasus WMS [134] that is equipped with a monitoring plugin as a part of the Panorama project [135]. The monitoring is done at a task level. Therefore, the measurements correspond to the independent execution of a task in a particular type of resource at a specified time.

The first phase of task runtime prediction extracts the pre-runtime configurations σ_i of a task t_i and the particular computational resource type v_i where the task will run. These are listed in Table 5.2. We decided to include the submission time (i.e., day and hour) to capture performance variability in clouds. For instance, a study by Jackson et al. [37] shows that the CPU performance of VMs in clouds was varied by 30% in compute time. Furthermore, Leitner and Cito [4] suggest that different running times may affect the performance of the cloud's resources.

Then, in the second phase, given the set of pre-runtime features σ_i of a task t_i, we estimated the time-series R_i for each metric defined in Table 5.1 using LSTM. The LSTM model is incrementally updated using data obtained after task t_h finishes executing. These data consist of its pre-runtime features σ_h and a set of resource consumption time-series R_h collected during the runtime. LSTM learns the resource consumption sequence per time step t and predicts the value of time step $t+1$ that are separated by time interval τ, and repeats the process until it reaches the desired time-series length n of time step $t+n$. Since every task has a different length of resource consumption record, we padded the end of the sequence with zeros until a specified length and removed the padded values at the end of the prediction. It needs to be noted that not all collected metrics have to be used in the prediction model as features. Feature selection can be made at this stage by calculating Pearson's correlation coefficient ρ of each metric to the actual task runtime [136].

The next step in the second phase is time-series feature extraction. The estimated time-series resource consumption R_i for a particular task t_i is pre-processed before being used as feature to estimate the task runtime in the third phase. We used time-reversal asymmetry statistic [137] to extract values ς_i from the estimated resource consumption R_i as shown in Eq. 5.1,

$$\varsigma_i(l) = \frac{\langle (x_{t+l} - x_t)^3 \rangle}{\langle (x_{t+l} - x_t)^2 \rangle^{\frac{3}{2}}} \tag{5.1}$$



(a) 1000 Genome workflow

(b) Autodock Vina workflow

Fig. 5.1: Sample of bioinformatics workflows

The feature that is extracted using this algorithm may represent the distinct time-series instance characteristics by calculating a value of specified sub-sequence with a window size determined by lag value l and performing surrogate data test $\langle . \rangle$ across the time-series. In a study by Fulcher and Jones [137], this technique has been proven to classify the time-series dataset of four classes using only one feature without error. Finally, in the third phase, the extracted relevant features ς_i for a task t_i from the second phase are combined with its pre-runtime features σ_i to predict the runtime.

5.6 Performance Evaluation

We evaluated the proposed approach with two bioinformatics workflows. The first workflow is based on the 1000 Genome Project[2], an international effort to establish a human genetic variation catalogue. Specifically, we used an existing 1000 Genome workflow[3] developed by Pegasus group to identify overlapping mutations. It provides a statistical evaluation of disease-related mutations. Its structure is shown in Fig. 5.1a. For our experiments, we analyzed the data corresponding to three chromosomes (chr20, chr21, chr22) of human genomes across five populations of African (AFR), Mixed American (AMR), East Asian (EAS), European (EUR), and South Asian (SAS).

The second workflow uses AutoDock Vina [138]–a molecular docking application–to screen a number of ligand libraries for plausible drug candidates (i.e., virtual screening). In particular, we used a virtual screening case of one receptor and forty ligands with various sizes and search spaces of the docking box taken from the Pegasus project[4]. The molecular docking tasks (i.e., AutoDock Vina) in this workflow can be considered as a bag of tasks where every task of receptor-ligand

[2] http://www.internationalgenome.org/about
[3] https://github.com/pegasus-isi/1000genome-workflow
[4] https://github.com/pegasus-isi/AutoDock-Vina-Workflow

Table. 5.3: NeCTAR virtual machines configuration

VM Type	vCPU	Memory	Storage	Operating System
m2.small	1	4GB	100GB	CentOS 7 (64-bit)
m2.medium	2	6GB	100GB	CentOS 7 (64-bit)
m2.large	4	12GB	100GB	CentOS 7 (64-bit)

docking can be executed in parallel before the compound selection task takes place to select the drug candidates. The structure of the virtual screening workflows is depicted in Fig. 5.1b.

To the best of our knowledge, this is the first work that predicts the runtime of workflow tasks using an online incremental learning approach. Hence, to compare our work with existing state-of-the-art solutions of task runtime prediction, we reproduced the batch offline learning work by da Silva et al. [35] that uses a task's input data as a feature to predict the task runtime. We refer to this approach as the baseline scenario. We also replicated the two-stage task runtime prediction in a batch offline learning methods by Pham et al. [36], which combined the input data, system configuration, and resource consumption to predict task runtime. We refer to this solution as the two-stage scenario. The latest solution is similar to our work except that we used the fine-grained resource consumption time-series data instead of an aggregated value of the consumed resources. We also implemented an online incremental version of both solutions to be compared with our proposed approach. To ensure the fairness of each evaluation, we used the IBk algorithm with default configuration for both batch offline and online incremental learning scenarios.

5.6.1 Experimental Setup

We set up the system on NeCTAR[5] cloud resources to evaluate the approaches. We used three different virtual machine types from NeCTAR, which are small, medium, and large flavours with eight, four, and two nodes configuration, respectively. They are configured to have the same storage capacity and operating system, as depicted in Table 5.3.

For the experiment, we generated between 900 and 12,000 executions for every task. The details of these tasks are depicted in Table 5.4. The resource consumption metrics for each running task are collected every time interval τ seconds, where $1 \leq \tau \leq 30$. Specifically, we used τ values of 1, 5, 10, 15, and 30 to analyze the trade-off between time-series granularity and learning

[5] https://nectar.org.au/

Table. 5.4: Summary of datasets

Workflow	**Task Name**	**Tasks per Workflow**	**Total Tasks Generated**
1000 Genome	individuals	10	9000
	individuals merge	1	900
	sifting	1	900
	mutation overlap	7	6300
	frequency	7	6300
Virtual Screening	autodock vina	40	12000
	compound selection	1	3000

performance. Furthermore, we defined the lag values l as $l = 2$ and $l = 3$ to see the effect of time-series data length on the feature extraction algorithm. The average runtime for task *individuals* is 158 seconds while *individuals merge* is 37 seconds. The shortest average runtime that can be monitored is 10 seconds for task *mutation overlap*, while *frequency* records 178 seconds on average and the *autodock vina* task shows the average runtime of 353 seconds. In this work, we did not consider the *sifting* task from the 1000 genome workflow and the *compound selection* task from the virtual screening workflow in our experiments since it has a very short runtime.

Regarding the learning algorithms, there are several configurable parameters for each of them. In general, we used the default configurations from their original implementation. It needs to be noted that we did not fine-tune the algorithms to get the optimal settings for this problem. Hence, further study to analyze the optimal configurations should be done as future work.

For the LSTM in resource consumption estimation, we used *batch size* $= 1$ since the system requires the data to be only seen once. Our LSTM implementation uses *sigmoid* as gates activation function, ten *hidden layers*, and one hundred *epochs* to train the model. Meanwhile, for IBk, we used the default parameter used by version 3.8 of the WEKA library where $k = 1$, no distance weighting, and linear function for the nearest neighbours search algorithm. For our batch offline learning experiments, we used various sizes of training data d to see the performance of classical batch offline learning related to the amount of data collection needed for building a good model for prediction. Specifically, we used the d values of 20%, 40%, 60%, and 80% in the experiments.

To validate the performance of our approach, we used relative absolute error (RAE) as a metric for evaluation as recomended in an empirical study by Armstrong et al. [139] over several alternative metrics as shown in Eq. 5.2,

$$RAE = \frac{\sum_{i=1}^{n} \mid r_{ij} - e_{ij} \mid}{\sum_{i=1}^{n} \mid r_{ij} - \frac{1}{n}\sum_{i=1}^{n} r_{ij} \mid} \tag{5.2}$$

where n is the number of predictions. The smaller the RAE value, the smaller the difference between the predicted value and the actual observed value.

5.7 Results and Analysis

In this section, we presented and analyzed the results of the experiments. We evaluated our proposed approach against the modified online incremental version of the baseline and two-stage scenarios. To ensure the fair comparison, we also presented the results of their original batch offline version approaches. Furthermore, we discussed the feature selection evaluation for our proposed method that can improve the model's accuracy.

5.7.1 Proposed Approach Evaluation

We evaluated our proposed approach with various time intervals τ, and time-series lags l. The value of time interval τ affects how often the system records the resource consumption of a particular task and impacts the length of the time-series data. Meanwhile, the value of the lag l that defines the time-reversal asymmetry statistics in feature extraction relies on the length of the time-series. Larger lag values may not be able to capture the distinctive profile of a short resource consumption time-series. Hence, we fine-tune these parameters for each task differently. The results of these experiments are depicted in Table 5.5; It is important to note that these do not include the feature selection mechanism in learning as we separate its evaluation in a different section.

In general, our proposed approach produced lower RAE compared to the baseline scenario and the two-stage scenario. From Fig. 5.2 we can see that exploiting fine-grained resource consumption significantly reduces the RAE of task runtime prediction for *individuals*, *individuals merge*, and *frequency*. Our proposed strategy showed a better result than baseline and two-stage for *mutation overlap* and *autodock vina*, although the difference is marginal. In this case, the fine-grained resource consumption features extracted using time-reversal asymmetry statistics may have a higher distinctive property that can characterize each instance uniquely compared to the aggregated value of resource consumption that is being used in the two-stage scenario.

Table. 5.5: Results of task estimation errors (RAE) using online incremental learning approach

Task	**Baseline**	**Two-Stages**	**Time-Series** ($l = 2$)				**Time-Series** ($l = 3$)			
			$\tau = 1s$	$\tau = 5s$	$\tau = 10s$	$\tau = 15s$	$\tau = 1s$	$\tau = 5s$	$\tau = 10s$	$\tau = 15s$
individuals	64.200	57.571	41.748	41.675	41.722	41.175	**39.180**	40.680	46.601	41.710
individuals merge	42.162	36.144	34.706	31.474	36.417	42.162	33.300	**29.553**	42.162	42.162
mutation overlap	5.778	3.615	**3.413**	3.682	5.729	5.778	3.861	3.949	5.778	5.778
frequency	48.971	37.327	35.523	31.039	30.812	32.251	35.386	**30.499**	35.108	32.368
autodock vina	5.380	5.153	4.170	4.062	**4.023**	4.081	4.140	4.045	4.049	4.090

Further analysis from Fig. 5.2 can explain the impact of lag values l on the performance. This graph shows the result from the baseline scenario, two-stage scenario, the best result from time-series $(l = 2)$, and $(l = 3)$ scenarios. As we can see the lag value $l = 3$ produces better results than $l = 2$ for all cases except *mutation overlap* and *autodock vina* with a marginal difference. In general, a higher lag value means a wider window size of the time-series to be inspected during the time-series feature extraction. However, if the length of the time-series is not long enough, the time-reversal asymmetry statistics cannot fully capture the distinctive characteristics of time-series instance. In *mutation overlap* case, many of the resource consumption time-series length is too short to be evaluated using the value of $l = 3$. Hence, the performance of the algorithm with a lag value of $l = 2$ achieves the lowest RAE. Meanwhile, for *autodock vina*, the trade-off between frequency measurement τ and lag l cannot be determined as the difference in error results in these various scenarios is insignificant.

Comprehensive results of the online incremental learning methods can be seen in Table 5.5. From the table, we can analyze the impact of configurable parameters to the algorithm's performance. *Individuals* achieves the lowest RAE for $l = 3$ and $\tau = 1s$. Moreover, *individuals merge* presents the best result for $l = 3$ and $\tau = 5s$ and *mutation overlap* shows the best result for $l = 2$ and $\tau = 1s$. Meanwhile, *frequency* gets the lowest RAE for $l = 3$ and $\tau = 5s$. Lastly, *autodock vina* achieves the lowest RAE for $l = 2$ and $\tau = 10s$. These results confirm our analysis from the previous discussion related to the length of the time-series record and the configurable parameters.

Furthermore, we can see that in several cases, the performance deteriorates to the value of baseline performance. This deterioration happens when the time-reversal asymmetry statistics cannot capture the time-series property because of the length limitation. Then the feature extrac-

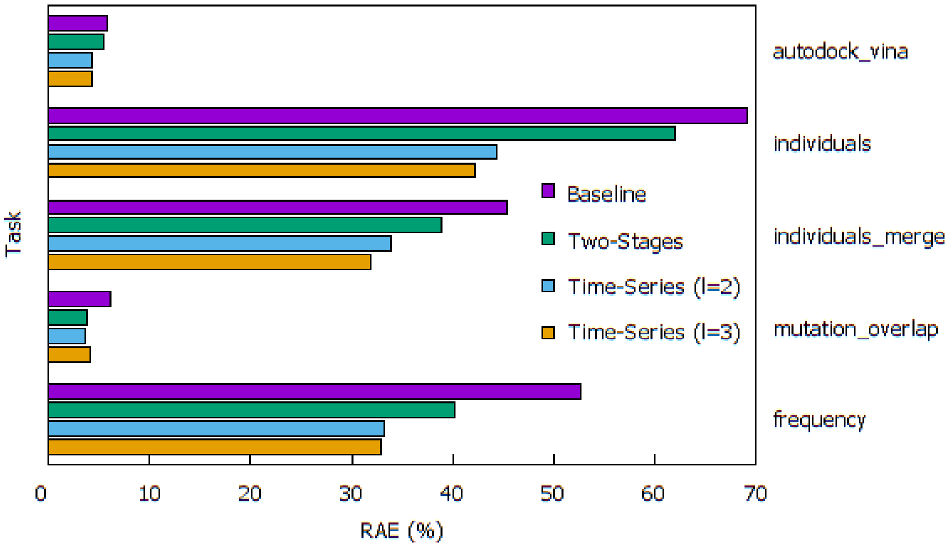


Fig. 5.2: Summary of task estimation errors (RAE) using online incremental learning approach

tion algorithm gives zero values. Hence, it produces the same result as the baseline scenario.

Therefore, the value of two configurable parameters in time-series feature extraction is an essential aspect in fine-tuning the prediction model. While in general, we can see that a higher lag l value produces a lower RAE, assigning an appropriate measurement interval τ must be further analyzed. There is no exact method to determine this frequency measurement value that is related to the prediction performance. The only known fact is that this value inflicts the size of time-series records to be stored in the monitoring database. We leave this problem as future work to improve the task runtime prediction method.

5.7.2 Batch Offline Evaluation

Since the original version of the baseline and two-stage scenarios are implemented in batch offline methods, we also evaluate these approaches to compare with their online incremental version. We use various sizes of training data d and test it using the rest of the data (i.e., $100\% - d$) for the baseline and two-stage scenarios. The result of these experiments is depicted in Table 5.6. In general, the performance of the prediction model improves as the size of data training increases. The results show the same trend for both baseline and two-stage scenarios, but it clearly shows that the two-stage outperform the baseline scenario for all cases. However, the performance of algorithms on more considerable data training becomes a trade-off to the temporal aspect that is

Table. 5.6: Results of task estimation errors (RAE) using batch offline learning approach

Task	Baseline				Two-Stages			
	$d = 20\%$	$d = 40\%$	$d = 60\%$	$d = 80\%$	$d = 20\%$	$d = 40\%$	$d = 60\%$	$d = 80\%$
individuals	65.543	62.587	64.523	66.049	59.117	57.625	57.113	**55.080**
individuals merge	42.522	42.256	37.936	37.424	37.177	35.227	34.312	**31.129**
mutation overlap	4.952	4.192	4.138	3.967	4.037	3.598	3.188	**2.936**
frequency	51.919	50.257	49.101	45.740	44.048	39.675	38.493	**36.622**
autodock vina	5.036	4.820	4.654	4.597	4.847	4.651	**4.529**	4.627

critical in WaaS platforms. This criticality is related to the juncture for collecting the data needed to build the model and the speed to compute the data training. Hence, more extensive data training may result in better algorithm performance but becomes a trade-off in terms of the processing speed. The results show the dependency of batch offline learning methods on the size of data collection for building a prediction model.

Furthermore, we also evaluated our online incremental learning version of the baseline and two-stage scenarios. The results of the online incremental learning approaches are inclusive in Table 5.5. For the baseline and two-stage scenario, the difference in a batch offline and online incremental learning is similar in all cases. We notice that the batch offline approaches outperform online incremental methods in most cases. However, it needs to be noted that such a result is gained after collecting–at least–40% data.

In the end, the improvement of task runtime prediction by using online incremental learning with time-series monitoring data is significant compared to the conventional batch offline learning methods that rely on the collection of data training beforehand to produce a good prediction model. This criticality limits the batch offline approach to be used in task runtime prediction for WaaS platforms. We argue that both practicality and performance results, the online incremental learning approach, may better suit the platform for task runtime prediction.

5.7.3 Feature Selection Evaluation

Further evaluation was done for the feature selection mechanism. We separate the evaluation to see the real impact of each feature on the learning performance. Hence, we consider the best scenarios from the previous experiment for this evaluation which are time-series scenario with

Table. 5.7: Results of Pearson's correlation based feature selection

Features	**individuals**	**individuals merge**	**mutation overlap**	**frequency**	**autodock vina**
stime	0.074	0.924	0.435	0.195	0.570
utime	0.003	0.060	0.995	0.935	0.974
iowait	0.216	0.053	0.006	0.121	-0.008
vmSize	0.027	-0.193	0.518	-0.112	-0.108
vmRSS	0.533	-0.255	0.946	-0.189	-0.129
read_bytes	0.004	0.322	0	-0.085	-0.278
write_bytes	0.187	-0.463	0.029	-0.237	-0.210
syscr	0.977	-0.608	-0.232	0.130	0.103
syscw	-0.810	-0.470	-0.153	-0.097	-0.582
rchar	0.981	-0.490	0.080	0.279	0.071
wchar	-0.032	-0.454	0.127	-0.212	-0.184
threads	-0.087	-0.103	-0.408	-0.052	0.019
procs	-0.087	-0.100	-0.412	-0.052	0.019

$l = 3$ and $\tau = 1s$ for *individuals*; $l = 3$ and $\tau = 5s$ for *individuals merge*; $l = 2$ and $\tau = 2$ for *mutation overlap*; and $l = 3$ and $\tau = 5s$ for *frequency*. In Table 5.7 we can see various correlation coefficient values for each feature for each task. A coefficient of zero means the feature is not correlated at all to the task runtime.

Meanwhile, a positive correlation value means there is a positive relationship between the feature and the runtime; as the feature value increases or decreases, the runtime follows the same trend. In this case, we select the features with $|\rho|$ values larger than a threshold and evaluate the performance of our approach. There is no exact rule on how to choose the limit. We choose the value based on small-scale experiments done beforehand, although it needs to be noticed that this value can easily be updated during runtime. Moreover, despite various features impacting differently for each task, CPU time (utime and stime), I/O system call (syscr and syscw), and I/O read (rchar) are the most common features that exceed the threshold.

From Fig. 5.3 we can see that feature selection impacts the task runtime prediction performance. Significant improvement can be observed for *individuals* and *frequency* with 6.49% and 3.49% error reductions respectively. *Individuals merge* show a slightly observed improvement of 0.59% while the improvement for *mutation overlap* is marginal with 0.04% error reduction. *Frequency* experiment uses $|\rho| = 0.5$ (two features). It only uses a small number of features to outperform the *without feature selection* scenario. Furthermore, *individuals* experiment uses $|\rho| = 0.4$ (six features), *mutation overlap* uses $|\rho| = 0.09$ (nine features), and the threshold for

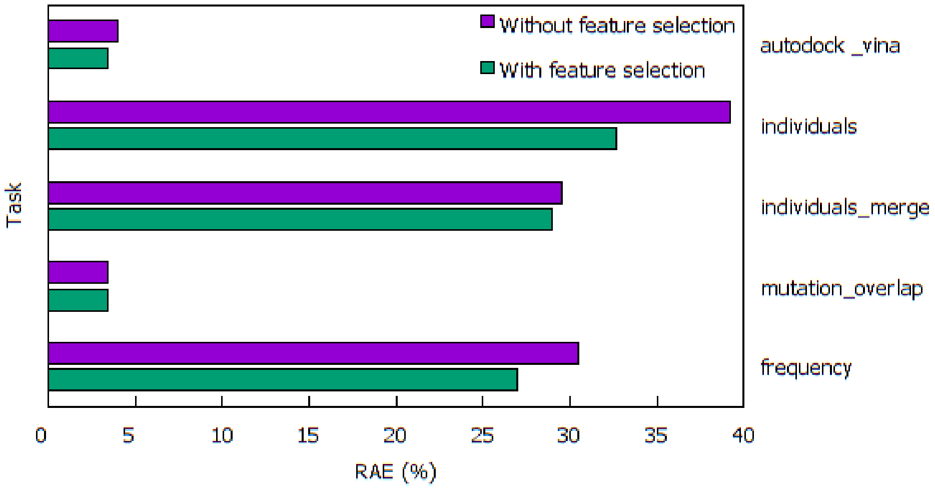


Fig. 5.3: Results of task estimation errors (RAE) with feature selection

individuals merge is $|\rho| = 0.08$ (nine features). The number of selected features are different for each task due to the difference in computational characteristics. The most distinctive features that represent the I/O intensive tasks are syscr and syscw. These features are observed in relatively high correlation value for task *individuals* and *individuals merge*. Meanwhile, the CPU intensive characteristics can be distinguished from high stime and utime feature correlation values as seen in task *mutation overlap*, *frequency*, and *autodock vina*.

5.8 Summary

In this chapter, we presented an online incremental approach for task runtime prediction of scientific workflows in cloud computing environments using time-series monitoring data. The problem of task runtime prediction is modelled based on the requirements for WaaS platforms, which offer the service to execute scientific workflows in cloud computing environments. Hence, approaches to task runtime prediction which use batch offline machine learning may not be suitable in this dynamic environment.

The strategy of using an online incremental learning approach is combined with the use of fine-grained resource consumption data in the form of time-series records such as CPU utilization and memory usage. We use a highly distinctive feature extraction technique called time-reversal asymmetry statistics that is capable of capturing the characteristics of a time-series record. Our proposal also considers the selection of features based on Pearson correlation to improve the task runtime prediction and to reduce the computational resources as the system only records the selected relevant features for all tasks.

From our experiments, the proposed approach outperforms baseline scenario and state-of-the-art methods in task runtime prediction. Although the variation of configurable parameters shows different results, in general, our proposal is better than previous solutions for task runtime prediction. Further result shows that our proposal achieves best-case and worst-case estimation errors of 3.38% and 32.69%, respectively. These results improve the performance, in terms of error, up to 29.89% compared to the state-of-the-art strategies.

As a part of future work, we plan to evaluate different machine learning algorithms, configurable parameters, and feature selection techniques that best suit specific task runtime prediction, since different settings of algorithms can result in different performance. Also, the variation of workflow tasks based on their sizes and computational characteristics such as data-intensive and compute-intensive tasks need to be explored to generate an effective strategy for enhancing the performance of the prediction model. Furthermore, the impact of cloud instances variability in the scheduling accuracy–which can affect the overall makespan–and the overhead performance impacting on the cost efficiency need to be analyzed more in-depth. This evaluation will help in demonstrating the eminence of the online incremental learning approach over the classical batch offline one.

This chapter presents the last works on theoretical strategies for scheduling multiple workflows in the WaaS platform in the thesis. Next, we extend the existing cloud workflow management system to handle multiple workflows deployment in cloud environments. This extension is an early step to build the WaaS platform that can serve the execution of scientific workflows in the clouds. We also implement the limited version of EBPSM algorithms in this system. This version of the algorithm is also equipped with one of the online incremental learning approaches that are presented in this chapter to predict the tasks' runtime that is integral to the scheduling processes.

Chapter 6

A System Prototype of the WaaS Platform

In this chapter, we extend the CloudBus WMS functionality to handle the workload of multiple workflows and develop a WaaS platform prototype. We implemented the ***E****lastic* ***B****udget-constrained resource* ***P****rovisioning and* ***S****cheduling algorithm for* ***M****ultiple workflows (EBPSM) for the platform and evaluated it using two bioinformatics workflows. Our experimental results show that the platform is capable of executing multiple workflows and minimizing their makespan while meeting the budget.*

6.1 Introduction

A conventional WMS is designed to manage the execution of a single workflow application. In this case, a WMS is tailored to a particular workflow application to ensure the efficient execution of the workflow. It is not uncommon for a WMS to be built by a group of researchers to deploy a specific application of their research projects. Developing the WaaS platform means scaling up the WMS functionality and minimizing any specific application-tailored in the component of the system. This challenge arises with several issues related to the resource provisioning and scheduling aspect of the WMS.

In this chapter, we focus on designing resource provisioning and scheduling module within the existing CloudBus WMS [9] for WaaS platform development. We modify the scheduling modules to fit the requirements by building the capability for scheduling multiple workflows. We develop the WaaS platform by extending CloudBus WMS functionality. We modify several components of the CloudBus WMS and implement the EBPSM algorithm that is designed to schedule budget-constrained multiple workflows to minimize the makespan while meeting the budget. The

This chapter is derived from: **Muhammad H. Hilman**, Maria A. Rodriguez, and Rajkumar Buyya. ‘Workflow-as-a-Service Cloud Platform and Deployment of Bioinformatics Workflow Applications.‘ *Knowledge Management in Development of Data-Intensive Software Systems (KMDDIS)*, 2020 (under review).

Table. 6.1: Summary of various WMS features

Main features		ASKALON	Galaxy	HyperFlow	Kepler	Pegasus	Taverna	CloudBus
Workflow Engine	Service-oriented	✓	✓	-	✓	✓	✓	✓
	GUI-supported	✓	✓	-	✓	✓	✓	✓
	Provenance-empowered	✓	✓	✓	✓	✓	-	✓
Distributed Environments	Grid-enabled	✓	✓	✓	✓	✓	✓	✓
	Cloud-enabled	✓	✓	✓	✓	✓	✓	✓
	Container-enabled	-	✓	✓	-	✓	-	-
	Serverless-enabled	-	-	✓	-	-	-	-

prototype is then evaluated using two bioinformatics workflows applications on various scenarios to demonstrate the system's capability to meet the WaaS platform requirements.

The rest of this chapter is organized as follows. Section 6.2 reviews works of that is related to our discussion. Section 6.3 describes the development of WaaS platform and its requirements. Furthermore, Section 6.4 explains the case studies of executing multiple workflows in WaaS platform. Finally, the Section 6.5 summarizes the findings.

6.2 Related Work

WMS technology has evolved since the era of cluster, grid, and current cloud computing environments. A number of widely used WMS were initially built by groups of multi-disciplinary researchers to deploy the life-science applications of their research projects developed based on the computational workflow model. Therefore, each of them has a characteristic tailored to their requirements. The case study of several prominent WMS is plentiful and worth to be explored further. The summary of these characteristics is depicted in table 6.1.

ASKALON [140] is a framework for development and runtime environments for scientific workflows built by a group from The University of Innsbruck, Austria. Along with ASKALON, the group released a novel workflow language standard developed based on the XML called Abstract Workflow Description Language (AWDL) [8]. ASKALON has a tailored implementation of wien2k workflow [141], a material science workflow for performing electronic structure calculations using density functional theory based-on the full-potential augmented plane-wave to be deployed within the Austrian Grid Computing network. Meanwhile, another project is Galaxy [142], a web-based platform that enables users to share workflow projects and provenance. It connects to myExperiments [143], a social network for sharing the workflow configuration and provenance among the scientific community. It is a prominent WMS and widely used for in silico experiments [144] [145] [146].

A lightweight WMS, HyperFlow [10] is a computational model, programming approach, and also a workflow engine for scientific workflows from AGH University of Science and Technology, Poland. It provides a simple declarative description based on JavaScript. HyperFlow supports the workflow deployment in container-based infrastructures such as docker and Kubernetes clusters. HyperFlow is also able to utilize the serverless architecture for deploying Montage workflow in AWS Lambda and Google Function, as reported by Malawski et al. [147]. Meanwhile, Kepler [148] is a WMS developed by a collaboration of universities, including UC Davis, UC Santa Barbara, and UC San Diego, United States. It is a WMS that is built on top of the data flow-oriented Ptolemy II system [149] from UC Berkeley. Kepler has been adopted in various scientific projects including the fluid dynamics [150] and computational biology [151]. This WMS provides compatibility to run on different platforms, including Windows, OSX, and Unix systems.

Another project is Pegasus [12], one of the prominent WMS that is widely adopted for projects that make an essential breakthrough to scientific discovery from The University of Southern California, United States. Pegasus runs the workflows on top of HTCondor [152] and supports the deployment across several distributed systems, including grid, cloud, and container-based environments. The Pegasus WMS has a contribution to the LIGO projects involved in the gravitational wave detection [153]. There is also Taverna [13], a WMS from The University of Manchester that as recently accepted under the Apache Incubator project. Taverna is designed to enable various deployment models from the standalone, server-based, portal, clusters, grids, to the cloud environments. Taverna has been used in various in silico bioinformatics projects, including several novel Metabolomics research [154] [155]. Finally, the CloudBus WMS [9], a cloud-enabled WMS from The University of Melbourne, is the center of discussion in this chapter. Its functionality evolves to support the development of the WaaS platform.

6.3 Prototype of WaaS Platform

In this section, we discuss a brief development of the CloudBus WMS and the WaaS platform development. The evolving functionality of CloudBus WMS in its first release to handle the deployment in the grid computing environment up to the latest version that provides the cloud-enabled functionality is described to give an overview of how the distributed systems change how the WMS works. Furthermore, we present the extension related to the scheduler component of this existing WMS to support the development of the WaaS platform.

6.3.1 CloudBus Workflow Management System

The earliest version of the WMS from the CLOUDS lab was designed for grid computing environments under the name of GridBus Workflow Enactment Engine in 2008. The core engine in this WMS was called a workflow enactment engine that orchestrated the whole workflow execution. The engine interacts with users through the portal that manages workflow composition and execution planning. This engine also equipped with the ability to interact with grid computing environments through the grid resource discovery to find the possible grid computational infrastructure, the dispatcher that sends the tasks to the grids for the execution, and the data movement to manage data transfer in and out through HTTP and GridFTP protocols. The Gridbus Workflow Enactment Engine was tested and evaluated using a case study of fMRI data analysis in the medical area. The architectural reference to this Gridbus Workflow Engine and its case study can be referred to the paper by Yu and Buyya [156].

The second version of the GridBus Workflow Enactment Engine was released in 2011, built with plugin support for deployment in cloud computing environments. In this version, the engine is equipped with the components that enable it to utilize several types of external computational resources, including grid and cloud environments. Therefore, it was renamed to CloudBus Workflow Engine. In addition to this functionality, the CloudBus Workflow Engine was tested and evaluated for scientific workflow execution on top of the Aneka Cloud Enterprise platform [157] and Amazon Elastic Compute Cloud (EC2) using a case study of evolutionary multiobjective optimization technique based on a genetic algorithm. We suggest that readers refer to the architectural design and case study implementation published by Pandey et al. [158].

The latest release of the CloudBus Workflow Engine in 2016 was the implementation of a comprehensive cloud-enabled functionality that allows the engine to lease the computational resources dynamically from the IaaS cloud providers. This version introduces a Cloud Resource Manager module that enables the platform to manage the resources (i.e., Virtual Machines) from several IaaS cloud providers related to its automated provisioning, integrating to the resource pool, and terminating the VMs based on the periodic scanning of the implemented algorithm. Along with

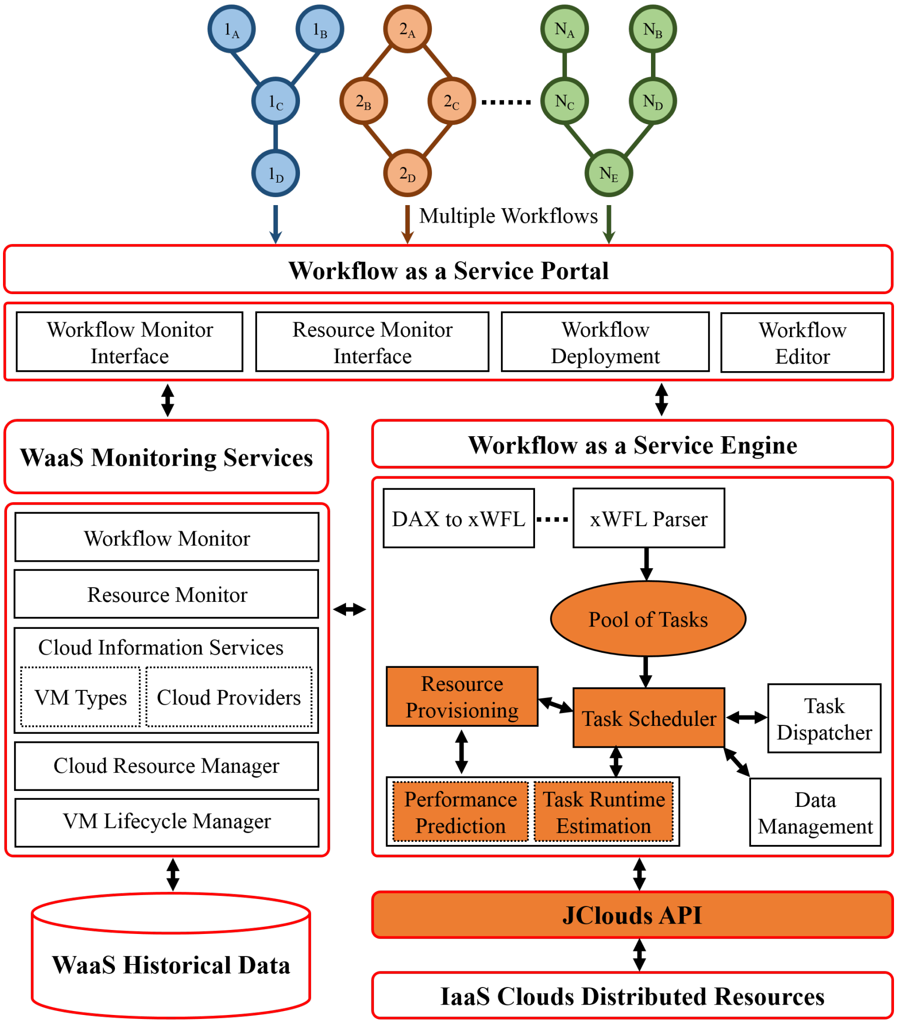


Fig. 6.1: Architectural reference on the WaaS platform

the dynamic functionality of cloud resources management, the WMS is also equipped with a dynamic algorithm to schedule workflows which is able to estimate the tasks' runtime based on the historical data from the previous workflows' execution. This version is known as the CloudBus Workflow Management System (CloudBus WMS). The architectural reference and its case study on Astronomical application Montage can be referred to the paper by Rodriguez and Buyya [9].

6.3.2 WaaS Platform Development

The CloudBus WMS is continuously adapting to the trends of the distributed systems infrastructures from cluster, grid, to the cloud environments. With the increasing popularity of the computational workflow model across scientific fields, we extend the CloudBus WMS to serve as a platform that provides the execution of workflow as a service. Therefore, we design the reference to the WaaS platform based on the structure of CloudBus WMS. Five entities compose the WaaS platform, they are portal, engine, monitoring service, historical database, and plugins to connect to distributed computing environments. This structure is similar to the previous CloudBus WMS architecture. The architectural reference is depicted in Fig. 6.1.

Portal: an entity that is responsible for bridging the WaaS platform to the users. The portal serves as the user interface in which users can submit the job, including composing, editing, and defining the workflow QoS requirements. It interacts with the engine to pass on the submitted workflows for scheduling. It also interacts with the monitoring service so that the users can monitor the progress of the workflows' deployment. Finally, the engine sends back the output data after it finished the execution through this entity. The change from the previous CloudBus WMS functionality is the capability of the portal to handle the workload of multiple workflows.

Monitoring Service: an entity that is responsible for monitoring the workflow execution and computational resources running within the WaaS platform that is provisioned from the cloud environments. Five components in this entity are the *Workflow Monitor* that tracks the execution of the jobs, the *Resource Monitor* which tracks the VMs running in the platform, the *Cloud Information Services* that discover the available VM types and images of the IaaS clouds profile, the *Cloud Resource Manager* that manages the provisioning of cloud resources, and the *VM Lifecycle Manager* which keeps tracking the VMs before deciding to terminate them.

This entity interacts with the portal to provide the monitoring information of workflows' execution. On the other hand, it also interacts with the engine to deliver the status of job execution for scheduling purposes and the computational resource availability status. We changed the provisioning algorithm, which is managed by the cloud resource manager and the VM lifecycle manager, based on the EBPSM algorithm. Both the cloud resource manager and the VM lifecycle manager control the VMs provisioning by keeping track of the idle status of each VM. They will be terminated if the idle time exceeded the $threshold_{idle}$. This provisioning algorithm is depicted in Chapter 4 in Algorithm 5. Finally, this entity saves the historical data of tasks' execution into the historical database where the information is used to estimate the task's runtime.

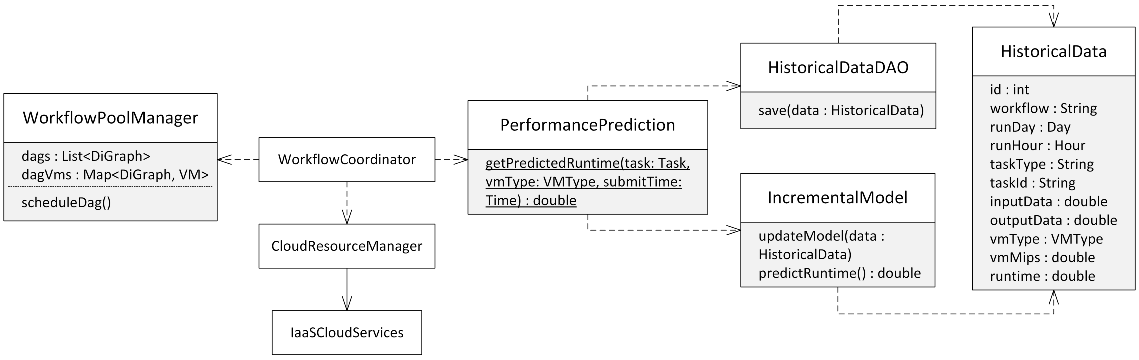


Fig. 6.2: Class diagram reference on Cloudbus Workflow Engine extension

Engine: an entity that is responsible for the orchestration of the whole execution of workflows. This entity interacts with the other objects of the WaaS platform, including the third party computational service outside the platform. Moreover, it takes the workflows' job from the portal and manages the execution of tasks. The scheduler that is part of this entity schedules each task from different workflows and allocates them to the available resources maintained by the monitoring service. It also sends the request to the plugins, JClouds API, for provisioning new resources if there is no available idle VMs to reuse.

Task scheduler, the core of the engine, is modified to adapt to the EBPSM algorithm that manages the scheduling of multiple workflows. Within the task scheduler, there is a component called the *WorkflowCoordinator* that creates the *Task Manager(s)* responsible for scheduling each task from the pool of tasks. To manage the arriving tasks from the portal, we create a new class *WorkflowPoolManager* responsible for periodically releasing the ready tasks for scheduling and keeping track of the ownership of each task.

Prediction component within the task scheduler is responsible for estimating the runtime of the task, which becomes a pre-requisite of the scheduling. We modify the *PredictRuntime* component to be capable of building an online incremental learning model. This learning model is a new approach for estimating the runtime for scientific workflows implemented in the WaaS platform. In the previous version, it utilizes statistical analysis to predict the tasks' runtime.

Historical database: an HSQL database used to store the historical data of tasks' execution. The information, then, is used to estimate the tasks' runtime. In this platform, we add the submission time variables to the database, since this information is used to build the prediction model to estimate the runtime.

Plugins: a JClouds API responsible for connecting the WaaS platform to third party computational resources. Currently, the platform can connect to several cloud providers, including Amazon Elastic Compute Cloud (EC2), Google Cloud Engine, Windows Azure, and OpenStack-based NeCTAR clouds. It sends the request to provision and terminates resources from the clouds.

Finally, the modified components within the WaaS platform from the previous version of the CloudBus WMS are marked with the red-filled diagram in Fig. 6.1 and the class diagram reference to the WaaS platform scheduler extension are depicted in Fig. 6.2.

6.3.3 Implementation of Multiple Workflows Scheduling Algorithm

Elastic **B**udget-constrained resource **P**rovisioning and **S**cheduling algorithm for **M**ultiple workflows, introduced in Chapter 4, is a dynamic heuristic algorithm designed for WaaS platform. The algorithm is designed to schedule tasks from multiple workflows driven by the budget to minimize the makespan. EBPSM distributes the workflow's budget to each of its tasks in the first step, and then, it manages the tasks from different workflows to schedule based on its readiness to run (i.e., parents' tasks finished the execution).

Furthermore, the algorithm looks for idle resources that can finish the task's execution as fast as possible without violating its assigned budget. This algorithm enforces the reuse of already provisioned resources (i.e., virtual machines) and sharing them between tasks from different workflows. This policy was endorsed to handle the uncertainties in the clouds, including VM performance variability, VM provisioning, and deprovisioning delays, and the network-related overhead that incurs within the environments. Whenever a task finishes, the algorithm redistributes the budget for the task's children based on the actual cost. In this way, the uncertainties, as mentioned earlier from cloud computing environments, can be further mitigated before creating a snowball effect for the following tasks.

The scheduling phase of the EBPSM algorithm was mainly implemented in the task scheduler, a part of the engine. The *WorkflowPoolManager* class receives the workflows' jobs and distributes the budget to the tasks as described in Chapter 3 in Algorithm 1. It keeps track of the workflows' tasks before placing the ready tasks on the priority queue based on the ascending Earliest Finish Time (EFT). Then, the *WorkflowCoordinator* creates a task manager for each task that is pooled from the queue. In the resource provisioning phase, the task scheduler interacts with the cloud resource manager in the monitoring resource to get the information of the available VMs. The task

Algorithm 7 Scheduling

```
procedure SCHEDULEQUEUEDTASKS($q$)
    sort $q$ by ascending Earliest Finish Time
    while $q$ is not empty do
        $t = q.poll$
        $vm = null$
        if there are idle VMs then
            $VM_{idle}$ = set of all idle VMs
            $vm = vm \in VM_{idle}$ that can finish $t$ within
                $t.budget$ with the fastest execution time
        else
            $vmt$ = fastest VM type within $t.budget$
            $vm = provisionVM(vmt)$
        $scheduleTask(t, vm)$
```

scheduler sends the request to provision a new VM if there are no VMs available to reuse. The implementation of this phase involves modules from different components of the WaaS platform. The scheduling algorithm is a simplified version of the original version introduced in Chapter 4. The detail of this implementation is depicted in Algorithm 7.

The post-scheduling of a task ensures budget preservation by calculating the actual cost and re-distributing the workflows' budget. This functionality was implemented in the task scheduler with additional information related to the clouds from the cloud information service, which maintains the cloud profile such as the VM types, and the cost of the billing period. The detail of the budget re-distribution procedure is described in Chapter 3 in Algorithm 3. In this work, we implemented a version of the EBPSM algorithm without the container. We did not need the container-enabled version as we only used bioinformatics workflow applications that did not have conflicting software dependencies and libraries. The enablement for microservices-supported WaaS platform is left for further development.

6.4 Case Studies and Performance Evaluation

In this section, we present the case study of multiple workflows execution within a WaaS platform prototype. We address the workload of bioinformatics workflows and its preparation for the execution. Furthermore, we also describe the technical infrastructure and its experimental design to deploy the platform and present the results from the experiment.

6.4.1 Bioinformatics Applications Workload

Many bioinformatics cases have adopted the workflow model for managing its scientific applications. An example is myExperiments [143] that has a broader scope to connect various bioinformatics workflows users. This social network for scientists who utilize the workflows for managing their experiments, stores almost four thousand workflows software, configurations, and datasets with more than ten thousand members. In this section, we explored two prominent bioinformatics workflows in the area of genomics analysis [159] and drug discovery [160] that had been extensively researched, as a case study for deploying multiple workflows in the clouds using the WaaS platform.

Identifying Mutational Overlapping Genes

The first bioinformatics case was based on the 1000 Genomes Project[1], an international collaboration project to build a human genetic variation catalogue. Specifically, we used an existing 1000 Genome workflow[2] to identify overlapping mutations in humans genes. The overlapping mutations were statistically calculated in a rigorous way to provide an analysis of possible disease-related mutations across human populations based on their genomics properties. This project has an impact on evolutionary biology. Examples include a project related to the discovery of full genealogical histories of DNA sequences [161].

1000 Genome workflow consists of five tasks that have different computational requirements [162]. They are *individuals*, *individuals_merge*, *sifting*, *mutations_overlap*, and *frequency*. *Individuals* performs data fetching and parsing of the 1000 genome project data that listed all Single Nucleotide Polymorphism (SNPs) variation in the chromosome. This activity involves a lot of I/O reading and writing system call. *Individuals_merge* showed similar properties, as it was a merging of *individuals* outputs that calculate different parts of chromosomes data. Furthermore, *sifting* calculates the SIFT scores of all SNPs variants. This task has a very short runtime. Finally, *mutations_overlap* calculates the overlapping mutations genes between a pair of individuals while *frequency* calculates the total frequency of overlapping mutations genes between several individuals. These two tasks are python-based application which requires intensive CPU and memory.

[1] http://www.internationalgenome.org/about

[2] https://github.com/pegasus-isi/1000genome-workflow

The 1000 Genome workflow takes two input, the chromosome data and its haplotype estimation (i.e., phasing) using shapeit method. The entry tasks were *individuals*, which extract each individuals from chromosome data, and *sifting* that calculates the SIFT scores from the phasing data. Furthermore, in the next level, *individuals_merge* merged all output from *individuals* and then, its output along with the *sifting* output becomes the input for the exit tasks of *mutation_overlap* and *frequency*. For our study, we analyzed the data corresponding to two chromosomes of chr21 and chr22 across five populations: African (AFR), Mixed American (AMR), East Asian (EAS), European (EUR), and South Asian (SAS). Furthermore, the structure of the 1000 Genome workflow was shown previously in Chapter 5 in Fig. 5.1a.

Virtual Screening for Drug Discovery

The second bioinformatics case used in this study was the virtual screening workflow. Virtual screening is a novel methodology that utilized several computational tools to screen a large number of molecules' libraries for possible drug candidates [163]. In simple terms, this (part of) drug discovery process involves two types of molecules, target receptors, and ligands that would become the candidates of drugs based on its binding affinity to the target receptor. This technique rises in popularity as the in-silico infrastructure and information technology are getting better. The virtual screening saves resources for in-vitro and in-vivo that require wet-lab experiments.

There are two main approaches in carrying out the virtual screening, ligand-based, and receptor-based virtual screening [164]. The ligand-based virtual screening relies on the similarity matching of ligands' libraries to the already known active ligand(s) properties. This activity is computationally cheaper than the other approach, as it depends only on the computation of the features of the molecules. On the other hand, the receptor-based virtual screening requires the calculation for both of the target receptors and the ligands to evaluate the possible interaction between them in a very intensive simulation and modelling. However, since the error rate of ligand-based virtual screening is relatively higher than the structure-based, this approach is applied as a filter step when the number of ligands involved in the experiments is quite high.

In this study, we used a virtual screening workflow using AutoDock Vina [138], a molecular docking application for structure-based virtual screening. In particular, we took a virtual screening case of one receptor and ligands with various sizes and search spaces of the docking box taken from

Table. 6.2: Various budgets used in evaluation

Name	B_1	B_2	B_3	B_4
1000 Genome Workflow				
chr21	US$ 0.1	US$ 0.25	US$ 0.45	US$ 0.65
chr22	US$ 0.1	US$ 0.25	US$ 0.45	US$ 0.65
Virtual Screening Workflow				
vina01	US$ 0.05	US$ 0.15	US$ 0.25	US$ 0.35
vina02	US$ 0.01	US$ 0.04	US$ 0.06	US$ 0.08

the Open Science Grid Project developed by the Pegasus group[3]. The receptor-ligand docking tasks in this workflow can be executed in parallel as in the bag of the tasks application model. Moreover, AutoDock Vina is a CPU-intensive application that can utilize the multi-CPU available in a machine to speed up the molecular docking execution. Therefore, two-level parallelism can be achieved to speed up the workflows, the parallel execution of several receptor-ligand docking tasks on different machines, and the multi-CPU parallel execution of a docking task within a machine. The structure of the virtual screening workflows was depicted previously in Chapter 5 in Fig. 5.1b.

Workload Preparation

The Pegasus group has developed the tools to generate both the 1000 Genome and Virtual Screening workflow based on the XML format. We converted the DAG generated from the tools into the xWFL, the format used by the WaaS platform. Based on this converted-DAG, we prepared two versions of the 1000 Genome workflows, which take two different chromosomes of chr21 and chr22 as input. Furthermore, we created two types of workflows that take as input two different sets of 7 ligands molecules for Virtual Screening.

We installed five applications for the 1000 Genome workflow in a custom VM image for the worker nodes. These applications are based on the Mutation_Sets project[4] and are available in the 1000 Genome workflow project[5]. It needs to be noted that the *mutation_overlap* and *frequency* tasks were python-based applications and have a dependency to the *python-numpy* and *python-matplotlib* modules. On the other hand, the only application that needs to be installed for the Virtual Screening workflow was AutoDock Vina[6], which can be installed without any conflict-

[3] https://github.com/pegasus-isi/AutoDock-Vina-Workflow

[4] https://github.com/rosafilgueira/Mutation_Sets

[5] https://github.com/pegasus-isi/1000genome-workflow

[6] http://vina.scripps.edu/

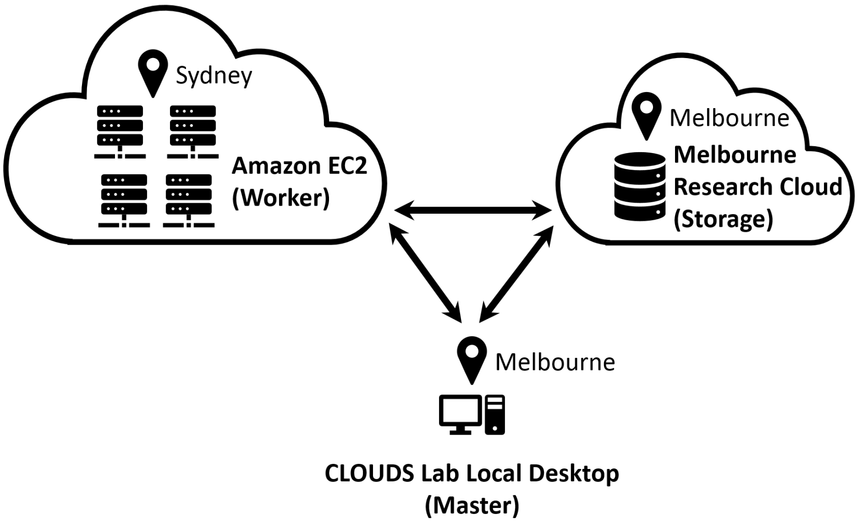


Fig. 6.3: Architectural reference on the WaaS platform nodes deployment

ing dependencies with the other workflow applications. Therefore, in this scenario, we did not encounter the conflicting dependencies problem.

We composed a workload that consists of 20 workflows with the types as mentioned earlier of applications that were randomly selected based on a uniform distribution. We also modelled four different arrival rates of those workflows based on a Poisson distribution from 0.5 workflows per minute (wf/m), which represents the infrequent requests, up to 12 wf/m that reflect the busiest hours. Each workflow was assigned a sufficient budget based on our initial deployment observation. We defined four different budgets for each workflow from B_1 to B_4, which represents the minimum to the maximum willingness of users to spend for particular workflows' execution. These budgets can be seen in Table 6.2.

6.4.2 Experimental Infrastructure Setup

Three components need to be deployed to ensure the running of the WaaS platform. The first is the master node containing the core of the workflow engine. This master node is the component that manages the lifecycle of workflows execution and responsible for the automated orchestration between every element within the platform. The second component is a storage node which stores all the data involved in the execution of the workflows. This storage manages the intermediate data produced between parents and children tasks' execution and acts as a central repository for the WaaS platform. Finally, the worker node(s) is the front runner(s) to execute the workflows'

Table. 6.3: Configuration of virtual machines used in evaluation

Name	vCPU	Memory	Price per second
CLOUDS Lab Local Desktop			
Master Node	4	8192 MB	N/A
Melbourne Research Cloud			
Storage Node	1	4096 MB	N/A
Amazon EC2			
Worker Node			
t2.micro	1	1024 MB	US$ 0.0000041
t2.small	1	2048 MB	US$ 0.0000082
t2.medium	2	4096 MB	US$ 0.0000164
t2.large	2	8192 MB	US$ 0.0000382

tasks submitted into the platform. The worker node(s) provisioning and lifespans are controlled based on the scheduling algorithms implemented in the core of the workflow engine.

For this experiment, we arranged these components on virtual machines with different configurations and setup. The master node was installed on Ubuntu 14.04.6 LTS virtual machine running in a local HP Laptop with Intel(R) Core(TM) i7-56000 CPU @ 2.60 GHz processor and 16.0 GB RAM. This virtual machine was launched using VMWare Workstation 15 player with 8.0 GB RAM and 60.0 GB hard disk storage. Moreover, we deployed the storage node on a cloud instance provided by The Melbourne Research Cloud[7] located in the *melbourne-qh2-uom* availability zone. This virtual machine was installed Ubuntu 14.04.6 LTS operating systems based on the *uom. general.1c4g* flavour with 1 vCPU, 4 GB RAM, and an additional 500 GB storage.

Furthermore, the worker node(s) were dynamically provisioned on Amazon Elastic Compute Cloud (EC2) Asia Pacific Sydney region using a custom prepared VM image equipped with the necessary software, dependencies, and libraries for executing 1000 Genome and Virtual Screening workflows. We used four different types and configurations for the worker nodes based on the family of T2 instances. The T2 instances family equipped with the high-frequency processors and have a balance of compute, memory, and network resources. Finally, the architectural reference for the nodes' deployment and its configuration are depicted in Fig. 6.3 and Table. 6.3 respectively.

[7] https://research.unimelb.edu.au/infrastructure/research-computing-services/services/research-cloud

6.4.3 Results and Analysis

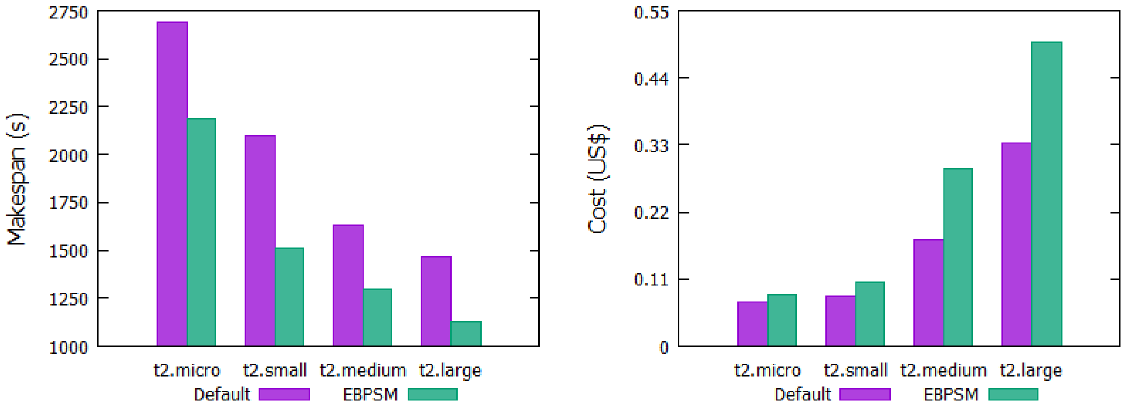
In this section, we present the comparison of EBPSM and First Come First Serve (FCFS) algorithm in a single workflow and homogeneous settings to ensure the fair evaluation. Then, it was followed by a thorough analysis of the EBPSM performance on a workload of multiple workflows in a heterogeneous environment represented by the different arrival rates of multiple workflows.

More Cost to Gain Faster Execution

The purpose of this experiment is to evaluate our proposed EBPSM algorithm for the WaaS platform compared to the default scheduler of the CloudBus WMS. This default scheduler algorithm did not rely on an estimate of tasks' runtime. It scheduled each task based on the first-come, first-served policy into a dedicated resource (i.e., VM) and terminated the resource when the particular task has finished the execution. Furthermore, this default scheduler was not equipped with the capability to select the resources in heterogeneous environments. Therefore, it only works for homogeneous cluster settings (i.e., clusters of one VM type only). Then, to have a fair comparison to the default scheduler, we modified the EBPSM algorithm to work for a single workflow in a homogeneous environment. We removed the module that enables EBPSM to select the fastest resources based on the task's sub-budget and let the algorithm provision a new VM if there are no idle VMs available to reuse, which means hiding the budget-driven ability of the algorithm.

In Fig. 6.4a, we can see that the homogeneous version of EBPSM was superior to the default scheduler on all scenarios. In this experiment, the default scheduler provisioned 26 VMs for each situation, while EBPSM only leased 14 VMs. In this case, we argue that the delays in initiating the VMs, which include the provisioning delay and delays in configuring the VM into the WaaS platform, have a significant impact on the total makespan. Therefore, the EBPSM can gain an average speedup of 1.3x faster compared to the default scheduler. However, this enhancement comes with a consequence of additional monetary cost.

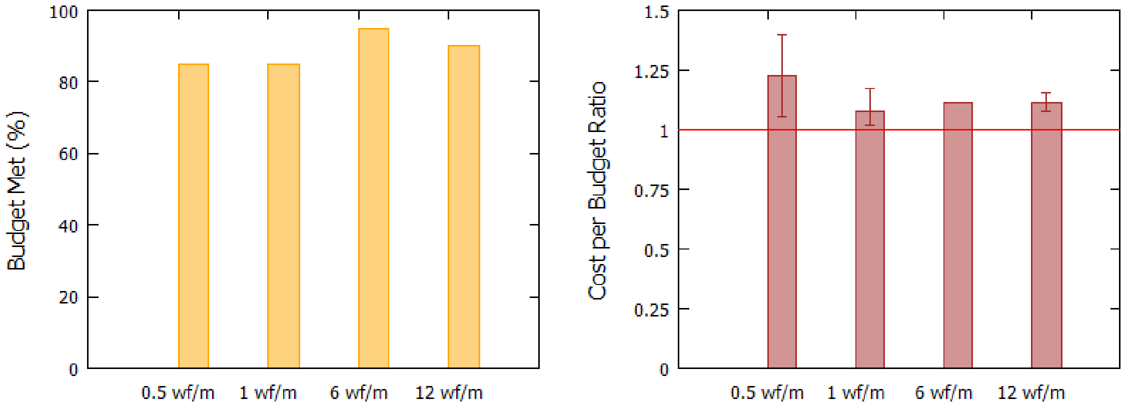
Fig. 6.4b showed that there is an increase in monetary cost for executing the workflows. The EBPSM lets the idle VM to become active for a certain period before being terminated, hoping that the next ready tasks would reuse it. This approach produced a higher cost compared to the immediate VM termination of the default scheduler approach. The average increase was 40% higher than the default scheduler. Is it worth to spend 40% more cost to gain 1.3x faster makespan? Further evaluation, such as Pareto analysis, needs to be done. However, more rapid responses



(a) Makespan

(b) Cost

Fig. 6.4: Makespan and cost of 1000 Genome (chr22) workflow on homogeneous environments



(a) Percentage of budget met

(b) Cost/budget ratio on budget violation cases

Fig. 6.5: Cost and budget analysis on workload with different arrival rate

to events such as modelling the storm, tsunami, and bush fires in the emergency situation, or predicting the cell location for critical surgery are undoubtedly worth more resources to be spent.

Budget Met Analysis

To evaluate the budget-constrained scheduling, we analyzed the performance of the EBPSM against its primary objective, meeting the budget. Two metrics were used in this analysis, the number of successful cases in meeting the budget, and the cost per budget ratio for any failed ones.

In this experiment, we observed the EBPSM performance in various arrival rate scenarios to see if this algorithm can handle the workload both in peak and non-peak hours. Fig. 6.5a showed that in the non-peak hours, the EBPSM could achieve 85% of the budget met while in the busier

Table. 6.4: Comparison of 1000 Genome (chr22) workflow in two environments

Name	Makespan (s)		Cost (US$)	
	Minimum	Maximum	Minimum	Maximum
Single - Homogeneous	2187	1125	0.084	0.499
Multiple - Heterogeneous	1819	1013	0.062	0.471

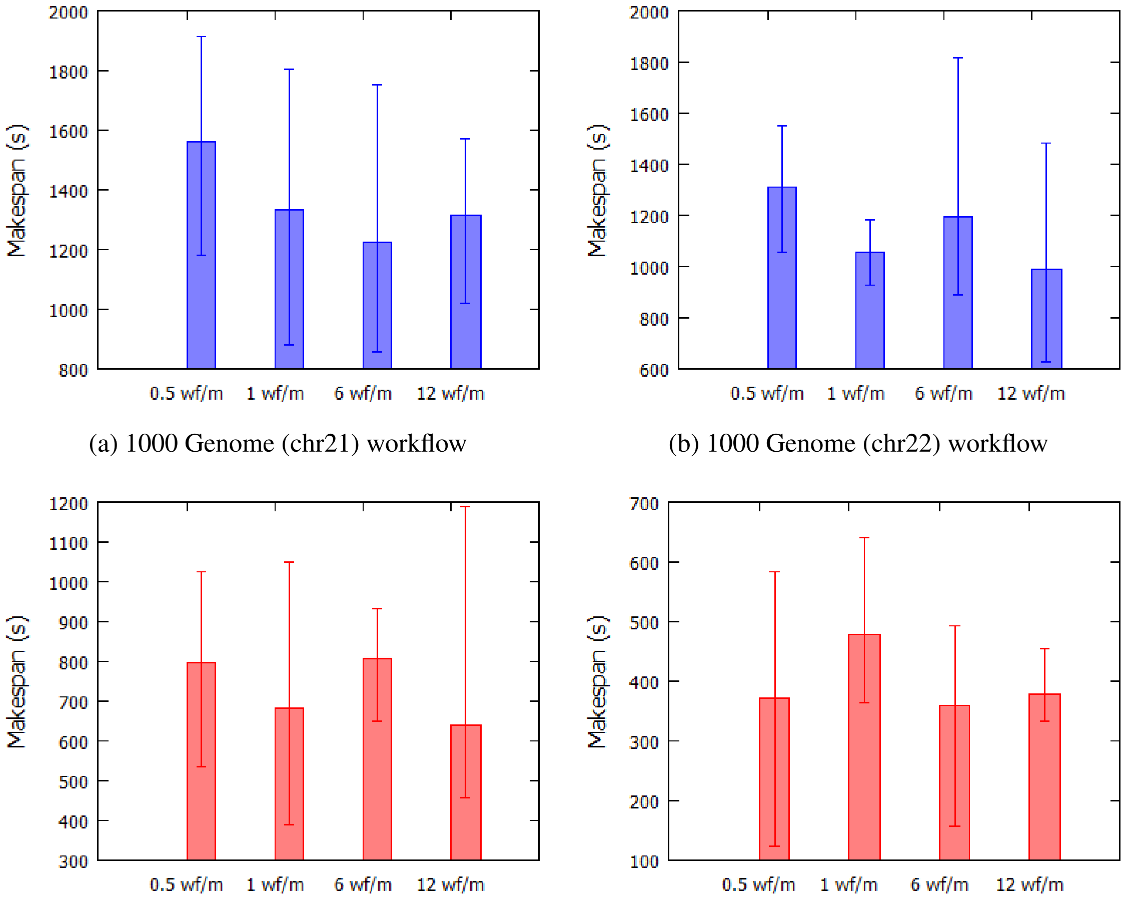
environment, this percentage increases up to 95%. In the peak-hours, there are more VMs to reuse and less idle time that makes the platform more efficient. However, it needs to be noted that there might exist some variability in the Amazon Elastic Compute Cloud (EC2) performance that might impact the results. Thus, the graphs did not show a linear convergence. Nevertheless, 85% of the budget-met percentage showed satisfactory performance for the algorithm.

The result of failed cases is depicted in Fig. 6.5b. From this figure, we can confirm the superiority of EBPSM for the peak-hours scenarios. The violation of the user-defined budget was not more than 15% in the peak-hours while the number increases up to 40% can be observed in the non-peak hours' settings. On average, the budget violation was never higher than 14% for all arrival rate schemes. Still and all, this violation was inevitable due to the performance variation of the Amazon Elastic Compute Cloud (EC2) resources.

Makespan Evaluation

It is essential to analyze the impact of scheduling multiple workflows on each of the workflows' makespan. We need to know whether sharing the resources between various users with different workflows is worth it and more efficient compared to a dedicated resource scenario in deploying the workflows. Before we discuss further, let us revisit the Fig. 6.4a, which showed the result of a single 1000 Genome (chr22) workflow execution in a homogeneous environment. Then, we compare it to the Fig 6.6b that presented the result for the same 1000 Genome (chr22) workflow in multiple workflows scenario and heterogeneous environment. If we zoom-in to the two figures, we could observe that EBPSM can further reduce both the makespan and the cost for the workflow in the latter scenario. We extracted these details of both scenarios into Table 6.4.

Let us continue the discussion for the makespan analysis. Fig. 6.6a, 6.6b, 6.6c, and 6.6d depicted the makespan results for 1000 Genome (chr21, chr22) and Virtual Screening (vina01, vina02) respectively. If we glance, there is no linear pattern showing the improvement of EBPSM



(a) 1000 Genome (chr21) workflow

(b) 1000 Genome (chr22) workflow

(c) Virtual Screening (vina01) workflow

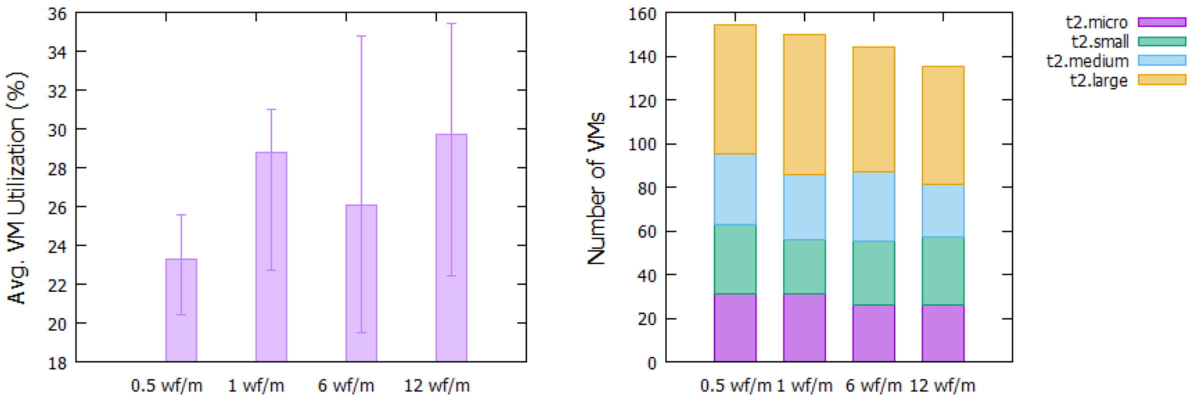
(d) Virtual Screening (vina02) workflow

Fig. 6.6: Makespan of workflows on workload with different arrival rate

performance over the different arrival rates of workflows. Nevertheless, if we observe further and split the view into two (i.e., peak hours and non-peak hours), we can see that the EBPSM, in general, produced better results for the peak-hour scenarios except for some outlier from 1000 Genome (chr22) and Virtual Screening (vina01) workflows. We thought that this might be caused by the number of experiments and the size of the workload. This is an important note to be taken as, due to the limited resources, we could not deploy workload with the scale of hundreds, even thousands of workflows.

VM Utilization Analysis

Finally, the last aspect to be evaluated regarding the EBPSM performance was VM utilization. It is the most important thing to be pointed out when discussing the policy of sharing and reusing



(a) Average VM utilization

(b) VM usage

Fig. 6.7: Average VM utilization and VM usage on workload with different arrival rate

computational resources. In Fig. 6.7a, we can see the increasing trend in VM utilization percentage along with the arrival rate of workflows on the platform. The average utilization upsurge for each scenario was 4%. The minimum utilization rate was 20% produced by the 0.5 wf/m scenario and the maximum of 36% for the 12 wf/m scenario.

We argue that the VM utilization rate had a connection to the number of VMs used during the execution. Fig. 6.7b depicted the number of VMs used in this experiment. We can observe that the overall number of VMs was declining along with the arrival rate of workflows. The average number of decrease was 20% for all VM types. The lowest drop was for the *t2.large* by 15%, and the highest drop was for the *t2.medium* by 25%. Meanwhile, the *t2.small* decreased by 22% and *t2.micro* by 16% respectively. The EBPSM algorithm always preferred to the fastest VM type and re-calculate and redistribute the budget after each task finished execution. Hence, in this case, the exit tasks might use more VMs of the cheapest type if the budget has been used up by the earlier tasks. Therefore, *t2.large* as the fastest VM type along with *t2.micro* as the cheapest would always be preferred compared to the other VM type.

From this experiment, we concluded that in the WaaS platform where the number of workflows involved is high, the scheduling algorithm must be able to maintain the low number of VMs being provisioned. Any additional VM leased means the higher possibility of incurring more delays related to the provisioning, initiating, and configuring the VMs before being allocated for executing the abundance of tasks.

6.5 Summary

The WMS have a crucial responsibility in executing scientific workflows. It manages the complicated orchestration process in scheduling the workflows and provisioning the required computational resources during the execution of scientific workflows. With the increasing trends of outsourcing computational power to third party cloud providers, there is a consideration to escalate the standalone execution of scientific workflows to the platform that provides the particular service. In this case, there is an emerging concept of a WaaS, extending the conventional WMS functionality to ensure the execution of scientific workflows as a utility service.

In this chapter, we extended the CloudBus WMS by modifying several components for it to be capable of scheduling multiple workflows to develop the WaaS platform. We implemented the EBPSM algorithm, budget-constrained scheduling algorithm designed for the WaaS platform that is capable of minimizing the makespan while meeting the budget. Furthermore, we evaluated the system prototype using two bioinformatics workflows applications with various scenarios. The experiment results demonstrate that the WaaS platform, along with the EBPSM algorithm, is capable of executing a workload of multiple bioinformatics workflows.

As this work primarily focused on designing the WaaS scheduler functionality, further development of the WaaS platform would be focused on developing the WaaS portal. It is the interface that connects the platform with the users. In this case, the users are expected to be able to compose and define their workflow's job, submit the job and the data needed, monitor the execution, retrieving the output from the workflow's execution. Finalizing the server-based functionality is another to-do list so that the WaaS platform can act as a fully functional service platform.

Finally, we plan to enable the WaaS platform for deploying workflows on microservices technology such as container technology, serverless computing, and unikernels system to accommodate the rising demand of the Internet of Things (IoT) workflows. This IoT demand is increasing along with the shifting from centralized infrastructure to distributed clouds. The shifting is manifested through the rising trends of edge and fog computing environments.

Chapter 7
Conclusions and Future Directions

7.1 Summary

Workflow is a computational model that represents a structured flow design process to manage the execution of large-scale applications. Therefore, these workflows require massive computational infrastructure and are deployed in distributed systems. Additionally, the rising trends in adopting the workflow in the scientific community and the increasing popularity of cloud computing create a demand to provide the execution of scientific workflows as a third party utility service. In this case, there emerges a new term for this platform called the Workflow-as-a-Service (WaaS) that provides the execution of workflow applications in cloud computing environments. This thesis addresses the problems of scheduling budget-constrained multiple workflows in the WaaS platform. The algorithms presented aim to overcome several challenges that inherently are characteristics of cloud computing environments where the WaaS platform is deployed.

Chapter 1 described the scope of this thesis by defining the problems and addressing the challenges in multiple workflows scheduling. It explained the backgrounds and the motivation that drives the research and summarizes the key contributions to address the issues. To understand the relevance of this research to the body of knowledge development in this field, Chapter 2 presented a taxonomy related to multiple workflows scheduling in multi-tenant distributed systems. It constructed the knowledge based on a thorough survey that compared and reviewed more than 31 algorithms published in peer-reviewed publications between 2008 to 2019.

Chapter 3 presented a task-based budget distribution strategy to assign the workflow's budget to individual tasks while considering the granularity of the cloud instances billing periods. This sub-budget then drives the resource provisioning and scheduling of the tasks, which aims to minimize the workflow's makespan while meeting the user-defined budgets. The algorithm incorporates the ability to handle several uncertainties, including performance variation and overhead delays in the cloud computing environments, by re-calculating the cost for executing a task and, whenever possible, re-distributing the remaining budget to the following tasks. The algorithm was evaluated using a simulation approach against a state-of-the-art level-based scheduling algorithm for scientific workflows called Budget Distribution with Trickling (BDT). The results show that our proposed algorithm has overall better performance than the BDT algorithm on various budget allocation scenarios.

Chapter 4 presented the **E**lastic **B**udget-constrained resource **P**rovisioning and **S**cheduling algorithm for **M**ultiple workflows (EBPSM) algorithm that is capable of minimizing makespan while meeting the budget. This algorithm incorporates the resource-sharing policy to reduce the overheads from acquiring, releasing the cloud resources, data transfer, and network-related activities. EBPSM optimizes the scheduling by reusing and sharing already provisioned VMs between tasks from different workflows, and only leased a new VM whenever there was no available idle VMs within the systems. The discussion in this chapter also described several versions of the EBPSM algorithm, which explore the different levels of sharing policy against various environment scenarios. The simulation experiment results show that the algorithm is capable of improving the performance along with the increasing frequency of arrival workflows into the WaaS platform. To the best of our knowledge, there are no other budget-constrained multiple workflows scheduling algorithms that aim to minimize the makespan while meeting the budget developed for the WaaS platform so far. In this case, the EBPSM was compared to a version of the budget-constrained algorithm called Minimizing the Schedule Length using the Budget Level (MSLBL) modified for scheduling multiple workflows. The results show that the EBPSM is capable of gaining further minimization of makespan while meeting the budgets in various scenarios.

Chapter 5 presented a novel approach in estimating the tasks' runtime based on the online incremental machine learning method. This estimation considers the uncertainties of cloud environments by comprehending the temporal aspects in the submission of the tasks into the learning model. This algorithm utilizes the time-series of resource consumption monitoring data as one of the essential features to predict the runtime. The evaluation is performed in a small-scale real-system experiment using a prominent workflow management system that is capable of retrieving the fine-grain monitoring data of the computational resources. The proposed algorithm was compared with the state-of-the-art batch offline learning approach to predict tasks' runtime for scientific workflows. To make a fair comparison, the batch offline learning method was modified to an online incremental learning version without the use of time-series resource consumption data. The results show that the proposed approach gains an overall better performance in various heterogeneous resources of cloud computing environments.

Chapter 6 presented the extension to the CloudBus WMS, a workflow management system designed for deploying scientific workflows in cloud computing environments. This extension supported the crucial functionality for the development of the WaaS platform, specifically related to the scheduling of multiple workflows, including the key features and its capability. The system prototype was demonstrated along with a case study using several bioinformatics workflows for genomic analysis and drug discovery and the implementation of the EBPSM algorithm.

7.2 Future Directions

Many challenges and problems in the current solutions should be considered for the future of scientific workflows, as nicely discussed in a study by Deelman et al. [165]. However, this thesis limited the discussion to the particular scheduling aspect in multi-tenant distributed computing systems. We captured the future direction of multi-tenancy from existing solutions and rising trend technologies that have a high potential to support the enhancement of multiple workflows scheduling. The range is broad from the pre-processing phase, which involves the strategy to accurately estimate task execution time that is a prerequisite for scheduling process–the scheduling techniques, that are aware of constraints such as failure, deadline, budget, energy usage, privacy, and security–to the use of heterogeneous distributed systems that differ not only in capacity but also pricing scheme and provisional procedures. Moreover, we observed a potential use of several technologies to enhance the multi-tenancy quality that comes from the rising trend technologies such as containers, serverless computing, Unikernels, and the broad adoption of the Internet of Things (IoT) workflows.

7.2.1 Advanced Multi-tenancy Using Microservices

Microservice is a variant of service-oriented architecture (SOA) that has a unique lightweight or even simple protocol and treated the application as a collection of loosely coupled service [166]. In this sense, we can consider container technology, serverless computing (i.e., function as a service), and Unikernels to fall into the category.

Kozhirbayev and Sinnott [118] reported that the performance of a container on a bare-metal machine is comparable to a native environment since no significant overhead is produced during the runtime. It is a promising technology to enhance multi-tenancy features for multiple workflows scheduling as it can be used as an isolated environment for workflow application before deploying it into virtual machines in clouds. We argue that, in the future, this technology will be widely used for solving multi-tenancy problem as it has been explored for executing a single scientific workflow as reported in several studies [111] [112] [113] [167].

However, the main trade-off of general-purpose container's performance (i.e., Docker) for scientific applications is the security [168]. The multi-tenancy requirements inevitably invite multiple users to share the same computational infrastructure at a time. In the case of Docker, every container process has access to Docker daemon, which is spawned as a child of the root. At any rate, this activity compromised the whole IT infrastructure. To tackle this security problem, Singularity Container that targets explicitly the scientific applications have been developed [169]. It has been tested in the Comet Supercomputer at the San Diego Supercomputer Center [170] and shown a promising result for handling multi-tenancy in the future WaaS platform. A study by Suhartanto et al. [171] showed that container could exploit multiple molecular docking processes within a host without significant performance degradation. In this way, the algorithms should be able to enhance the multi-tenancy of a host VM by deploying multiple jobs within acceptable deterioration. However, the algorithms must consider the additional container initiating delay [121] as a part of their design.

Another promising technology is serverless computing/Function-as-a-Service (FaaS). FaaS is a new terminology that stands on the top of cloud computing as a simplified version of the virtualization service. In this way, cloud providers directly manage resource allocation, and the users only needed to pay for the resource usage based on the application codes. This technology facilitates the users who need to run specific tasks from a piece of code without having a headache in managing the resources in cloud computing environments. We consider to include this into the future directions since the high potential of its multi-tenancy service to accommodate the multiple workflows scheduling.

Furthermore, this technology has been tested for single scientific workflow execution, as reported by Malawski [172], Jiang et al. [173], and Malawski et al. [147]. Notably, this FaaS can serve the workloads that consist of platform-independent workflows, which can be efficiently executed on top of this facility without having to provision a new virtual machine. Nevertheless, deploying scientific workflows in the serverless infrastructure is limited by the size of applications' requirements in terms of CPU, memory, and network, as reported by Spillner et al. [174].

Finally, Unikernels is another virtualization technology that is designed to maintain perfect isolation of virtual machines and maintain the lightweight of the container [175]. Unikernels enhance the virtualization in terms of weight by removing a general-purpose OS from the VM. In this way, Unikernels directly run the application on the virtual hardware. Even more impressive, recent finding shows that Unikernels do not require a virtual hardware abstraction. It can directly run as a process by using an existing kernel system that is called whitelisting mechanism [176]. Looking into the combination features of virtual machines and containers in one single virtualization technology, we can hope for a better multi-tenancy service for the WaaS platform.

7.2.2 Reserved vs On-demand vs Spot Instances

The further reduction of operational cost has been a long-existing issue in utility-based multi-tenant distributed computing systems. Notably, in cloud computing environments where the resources are leased from third-party providers based on various pricing schemes, cost-aware scheduling strategy is highly considered. Most of the algorithms for clouds use on-demand instances which ensure the reliability in a pay-as-you-go pricing model. Meanwhile, a work by Zhou et al. [44] explored the use of spot instances that is cheaper, but less reliable as they were limited time available and could be terminated at an unpredictable time. This type of resource raises a fault-tolerant issue to be considered.

On the other hand, the use of reserved instances in clouds should be explored to minimize further the total operational cost as the pricing of this bulk reservation is lower than on-demand even spot instances. The issue of using reserved instances is related to how accurate the algorithms can predict the workload of workflows to lease a number of reserved instances. The combination of reserved, on-demand, and spot instances must be explored to create an efficient resource provisioning strategy. The workload pattern forecasting model also should accompany the algorithms to better provision the resources.

7.2.3 Multi-clouds vs Hybrid Clouds vs Bare-metal Clouds

The use of multi-cloud computing providers for scientific workflow was explored by Jrad et al. [177] and Montes et al. [178] by introducing algorithms that were aware of different services available. However, the only relevant works found in our study are the use of hybrid clouds for separating tasks execution. In our survey, a work used hybrid clouds to treat tasks with different privacy levels in healthcare services, while another research utilized public clouds to cover the computational need that could not be fulfilled using private clouds and on-premises infrastructure. In our opinion, further utilization of multi-clouds can be beneficial as a single cloud provider may not be able to serve the high requirements of resources in WaaS platforms on particular peak hours. The other advantage of multi-clouds is the possible reduction of operational cost. In this way, discovering relevant services can be further explored by choosing a particular data center location to minimize data movements. This approach also may consider comparing the ratio of price and performance from various cloud instances from multiple cloud providers, as various cloud providers charge different prices for the datacenter in different geographical locations.

In this resource heterogeneity discussion, we have to mention a valuable service that provides a more heterogeneous infrastructure that is called bare-metal clouds. Bare-metal cloud is an emerging service in the IaaS business model that leases a physical machine instead of a virtual machine to the users. This service targets user that need specific hardware requirements in an intensive computation (i.e., GPU, FPGA). While one may ask the elasticity of this service against any standard cloud services, recent work has shown that such agility in provisioning bare-metal clouds can be compared to general VM virtualization [179]. On the other hand, the challenge of managing such an environment must be considered when designing the algorithms. In this way, the scheduling policy should calculate the several possible overhead factors (i.e., network bandwidth, end-to-end latency) in comparison to the monetary cost of the infrastructure.

7.2.4 Fast and Reliable Task Runtime Estimation

Predicting task runtime in clouds is non-trivial, mainly due to the problem in which cloud resources are subject to performance variability [37]. This variability occurs due to several factors–including virtualization overhead, multi-tenancy, geographical distribution, and temporal aspects [4]–that affect not only computational performance, but also the communication network used to transfer input/output data [119]. The majority of algorithms rely on the estimation of task execution time to produce an accurate schedule plan. Meanwhile, the works on task runtime estimation in scientific workflows are limited, including the latest works by Pham et al. [36] and Nadeem et al. [180] that used machine learning techniques. Previously, work on scientific workflow profiling and characterization by Juve et al. [27] that produced a synthetic workflow generator was being used by the majority of works on workflow scheduling.

The future techniques must be able to address dynamic workloads that are continuously arriving in resemblance to stream data processing. The adoption of an online and incremental machine learning approach may become another solution. In this approach, the algorithm does not need to learn from a model constructed from a large number of collected datasets, which is generally time-consuming and compute-intensive. The algorithm only sees the data once and then integrates additional information as the model incrementally built from new data. The latest work by Sahoo et al. [181] develops OMKR, an online and incremental machine learning approach, to handle large time-series datasets in a near real-time process. While this approach is still intensively being studied for scientific workflow area, the preliminary work has been discussed in Chapter 5 for future WaaS platform. These methods should mainly consider the concept drift that impacted the performance variability of the clouds as reported for other continuous data streams problem [182].

7.2.5 Integrated Anomaly Detection and Fault-tolerant Aware Platforms

Detecting anomaly in scientific workflows is one of the challenges to maintain the fault-tolerance of scheduling in cloud computing environments. Several notable examples include a work by Samak et al. [183] that detailed integrated workflows and resource monitoring for the STAMPEDE project. Furthermore, Gaikwad et al. [184] used Autoregression techniques to detect the anomalies by monitoring the systems and a similar work by Rodriguez et al. [185] adopted Neural Network methods. On the other hand, the fault-tolerant algorithms found in our survey used replication technique [76] and checkpointing [44] to handle failure in workflows execution.

Future works on this area include the integration of detecting anomalies and failure-aware scheduling in multi-tenant computing platforms and the use of various fault-tolerant methods in failure-aware algorithms, such as resubmission and live migration. Furthermore, to investigate how the anomalies detection model can be combined with the task runtime prediction model to better schedule multiple tasks in heterogeneous environments also require attention. In this case, the algorithms should incorporate the ability to be fully aware of the underlying hardware performance and their monitoring features. The algorithms then can decide to either resubmit the anomalous jobs or duplicated the jobs in the first place to ensure the completion of the workflows.

7.2.6 Multi-objective Constraints Scheduling

The flexibility and ability to easily scale the number of resources (i.e., VMs) in the cloud computing environments leads to a trade-off between two conflicting QoS requirements: time and cost. In this case, the more powerful VMs capable of processing a task faster will be more expensive than the slower, less powerful ones. There has been an extensive research [26] on this scheduling topic that specifically designed for cloud computing environments, with most works proposed algorithms that were aimed to minimize the total execution cost while finishing the workflow execution before a user-defined deadline. Meanwhile, the works that aimed to minimize the makespan by fully utilizing the available budget to lease faster resources, as much as possible, are limited. We identified algorithms that considered budget in their scheduling such as [56], [94], [186], and [187] that exploited the budget as a complementary constraint to the deadline. Nevertheless, none of them aims to fully utilize the available budget to get a faster execution time.

Furthermore, studies on multiple workflows scheduling that aims explicitly to achieve multi-objective optimization (i.e., time and cost minimization) are also minimal. While the metaheuristics and evolutionary programming have been used, such as the work by Fard et al. [188], its implementation for multiple workflows scheduling is limited by its pre-processing requirement. However, a more lightweight list-based heuristic approach such as MOHEFT [84] and DPDS [189] can be considered for multiple workflows scheduling. In this case, the algorithms should carefully handle the trade-off between achieving two or more objective constraints and maintain the lightweight low-complexity scheduling process. The lightweight scheduling can be achieved by exploiting heuristics approaches to gain a relatively good optimization instead of aiming for the optimal schedule through sophisticated methods with the expensive computational cost such as evolutionary and bio-inspired computing algorithms.

7.2.7 Energy-efficient Computing

Beloglazov et al. [190] have extensively explored the issue of green computing in cloud datacenters. Interestingly, there are several works in our study that addressed this energy-efficient and carbon footprint issue. While a work discussed energy-efficient strategy at the infrastructure level by implementing a live migration technique [81], another work tackled it at workload level by allocating the load to specific physical machines [73].

For the WaaS platform providers that rely on IaaS clouds to lease the computational resources, adopting workload level strategies for energy-aware scheduling is one of the possible further explorations. In this case, they do not have direct control over raw computational infrastructures as IaaS cloud providers do. Therefore, the algorithms should consider the energy-aware strategy of choosing the green computational resources, as discussed by Toosi et al. [191] in a work that explored a renewable-aware geographical load balancing.

7.2.8 Privacy-aware Scheduling

The users' privacy is an essential aspect that has been tackled by separating the execution in a private and public cloud-based on their data privacy level [63]. However, it is crucial to consider the security aspect of managing privacy since both issues are highly inter-related. One of the works that considers security is the SABA algorithm [192]. However, it is designed for a single workflow scheduling and intended to explore the relationship between cost and security aspects in the scheduling, instead of focusing on privacy.

Further exploration of privacy and security in the multiple workflows scheduling has to be elaborated as it resembles the real world workflow application problems. Another way to deal with confidentiality is by adopting a reliable security protocol for data processing in cloud computing environments, such as homomorphic encryption [193]. However, one inevitable trade-off from the attempt to increase the security aspect is an additional delay to the total makespan. The multiple workflows scheduling algorithms should consider this trade-off and include it as a part of the scheduling strategy design.

7.2.9 Internet of Things (IoT) Workflows

A visionary paper by Gubbi et al. [194] mentions a future use of the Internet of Things (IoT) in a workflow form. The idea has been implemented in several works, including a smart city system [195] and a big data framework [196]. This type of workflow increases in numbers, and its broad adoption is predicted to be widely seen shortly. Therefore, the need for a multi-tenant computing platform that can handle such workflows may arise.

IoT applications are highly demanding network resources to handle end-to-end services from sensors to users. Therefore, network-intensive strategies such as bandwidth-aware and latency-aware must be considered in the scheduling. A recent study by Stavrinides and Karatza [197] presents a work that is aware of edge and cloud resources available to differentiate the task allocation based on their computational requirements.

Bibliography

[1] R. Barga and D. Gannon, *Scientific versus Business Workflows - Workflows for e-Science: Scientific Workflows for Grids*. London: Springer London, 2007, pp. 9–16.

[2] D. Gannon, E. Deelman, M. Shields, and I. Taylor, *Introduction - Workflows for e-Science: Scientific Workflows for Grids*. London: Springer London, 2007, pp. 1–8.

[3] C. Goncalves, L. Assuncao, and J. C. Cunha, "Data Analytics in The Cloud with Flexible MapReduce Workflows," in *Proceedings of The 4th IEEE International Conference on Cloud Computing Technology and Science*, Dec. 2012, pp. 427–434.

[4] P. Leitner and J. Cito, "Patterns in the Chaos: A Study of Performance Variation and Predictability in Public IaaS Clouds," *ACM Transactions on Internet Technology*, vol. 16, no. 3, Apr. 2016.

[5] Y. Xing and Y. Zhan, "Virtualization and Cloud Computing," in *Future Wireless Networks and Information Systems*, Y. Zhang, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 305–312.

[6] C. Wu, R. Buyya, and K. Ramamohanarao, "Cloud Pricing Models: Taxonomy, Survey, and Interdisciplinary Challenges," *ACM Computing Surveys*, vol. 52, no. 6, Oct. 2019.

[7] K. Svitil, "Gravitational Waves Detected 100 Years After Einstein's Prediction," 2016. [Online]. Available: http://www.caltech.edu/news/gravitational-waves-detected-100-years-after-einstein-s-prediction-49777

[8] J. Qin and T. Fahringer, *Scientific Workflows: Programming, Optimization, and Synthesis with ASKALON and AWDL*. Springer Publishing Company, Incorporated, 2014.

[9] M. A. Rodriguez and R. Buyya, "Scientific Workflow Management System for Clouds," in *Software Architecture for Big Data and the Cloud*, I. Mistrik, R. Bahsoon, N. Ali, M. Heisel, and B. Maxim, Eds. Boston: Morgan Kaufmann, 2017, pp. 367–387.

[10] B. Balis, "HyperFlow: A Model of Computation, Programming Approach and Enactment Engine for Complex Distributed Workflows," *Future Generation Computer Systems*, vol. 55, pp. 147–162, 2016.

[11] P. Korambath, J. Wang, A. Kumar, L. Hochstein, B. Schott, R. Graybill, M. Baldea, and J. Davis, "Deploying Kepler Workflows as Services on a Cloud Infrastructure for Smart Manufacturing," *Procedia Computer Science*, vol. 29, pp. 2254–2259, 2014, 2014 International Conference on Computational Science.

[12] E. Deelman, K. Vahi, M. Rynge, R. Mayani, R. F. da Silva, G. Papadimitriou, and M. Livny, "The Evolution of the Pegasus Workflow Management Software," *Computing in Science Engineering*, vol. 21, no. 4, pp. 22–36, July 2019.

[13] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. Nieva de la Hidalga, M. P. Balcazar Vargas, S. Sufi, and C. Goble, "The Taverna Workflow Suite: Designing and Executing Workflows of Web Services on the Desktop, Web or in the Cloud," *Nucleic Acids Research*, vol. 41, no. 1, pp. 557–561, 05 2013.

[14] B. Howe, G. Cole, E. Souroush, P. Koutris, A. Key, N. Khoussainova, and L. Battle, "Database-as-a-Service for Long-Tail Science," in *Scientific and Statistical Database Management*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 480–489.

[15] J. Wang, P. Korambath, I. Altintas, J. Davis, and D. Crawl, "Workflow as a Service in The Cloud: Architecture and Scheduling Algorithms," *Procedia Computer Science*, vol. 29, pp. 546–556, 2014, 2014 International Conference on Computational Science.

[16] S. Esteves and L. Veiga, "WaaS: Workflow-as-a-Service for The Cloud with Scheduling of Continuous and Data-intensive Workflows," *The Computer Journal*, vol. 59, no. 3, pp. 371–383, March 2016.

[17] B. P. Rimal and M. Maier, "Workflow Scheduling in Multi-tenant Cloud Computing Environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 1, pp. 290–304, Jan. 2017.

[18] "Gartner Forecasts Worldwide Public Cloud Revenue to Grow 172020," Nov 2019. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2019-11-13-gartner-forecasts-worldwide-public-cloud-revenue-to-grow-17-percent-in-2020

[19] S. G. Ahmad, C. S. Liew, M. M. Rafique, E. U. Munir, and S. U. Khan, "Data-Intensive Workflow Optimization Based on Application Task Graph Partitioning in Heterogeneous Computing Systems," in *Proceedings of The 4th IEEE International Conference on Big Data and Cloud Computing*, Dec. 2014, pp. 129–136.

[20] J. Yu and R. Buyya, "A Taxonomy of Workflow Management Systems for Grid Computing," *Journal of Grid Computing*, vol. 3, no. 3, pp. 171–200, Sep. 2005.

[21] M. Wieczorek, A. Hoheisel, and R. Prodan, *Taxonomies of the Multi-Criteria Grid Workflow Scheduling Problem*. Boston, MA: Springer US, 2008, pp. 237–264.

[22] F. Wu, Q. Wu, and Y. Tan, "Workflow Scheduling in Cloud: A Survey," *The Journal of Supercomputing*, vol. 71, no. 9, pp. 3373–3418, Sep. 2015.

[23] S. Singh and I. Chana, "A Survey on Resource Scheduling in Cloud Computing: Issues and Challenges," *Journal of Grid Computing*, vol. 14, no. 2, pp. 217–264, Jun. 2016.

[24] E. N. Alkhanak, S. P. Lee, and S. U. R. Khan, "Cost-aware Challenges for Workflow Scheduling Approaches in Cloud Computing Environments: Taxonomy and Opportunities," *Future Generation Computer Systems*, vol. 50, pp. 3–21, 2015, Quality of Service in Grid and Cloud 2015.

[25] S. Smanchat and K. Viriyapant, "Taxonomies of Workflow Scheduling Problem and Techniques in The Cloud," *Future Generation Computer Systems*, vol. 52, pp. 1–12, 2015, Special Section: Cloud Computing: Security, Privacy and Practice.

[26] M. A. Rodriguez and R. Buyya, "A Taxonomy and Survey on Scheduling Algorithms for Scientific Workflows in IaaS Cloud Computing Environments," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 8, p. e4041, 2017, e4041 cpe.4041.

[27] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and Profiling Scientific Workflows," *Future Generation Computer Systems*, vol. 29, no. 3, pp. 682–692, 2013, Special Section: Recent Developments in High Performance Computing and Security.

[28] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The Cost of Doing Science on The Cloud: The Montage Example," in *Proceedings of The ACM/IEEE Conference on Supercomputing*, Nov. 2008, pp. 1–12.

[29] P. Maechling, E. Deelman, L. Zhao, R. Graves, G. Mehta, N. Gupta, J. Mehringer, C. Kesselman, S. Callaghan, D. Okaya, H. Francoeur, V. Gupta, Y. Cui, K. Vahi, T. Jordan, and E. Field, *SCEC CyberShake Workflows–Automating Probabilistic Seismic Hazard Analysis Calculations*. London: Springer London, 2007, pp. 143–163.

[30] P. Nguyen and K. Nahrstedt, "MONAD: Self-adaptive Micro-service Infrastructure for Heterogeneous Scientific Workflows," in *Proceedings of The 2017 IEEE International Conference on Autonomic Computing*, July 2017, pp. 187–196.

[31] L. Bryant, J. Van, B. Riedel, R. W. Gardner, J. C. Bejar, J. Hover, B. Tovar, K. Hurtado, and D. Thain, "VC3: A Virtual Cluster Service for Community Computation," in *Proceedings of the Practice and Experience on Advanced Research Computing*, ser. PEARC '18. New York, NY, USA: Association for Computing Machinery, 2018.

[32] M. Belkin, R. Haas, G. W. Arnold, H. W. Leong, E. A. Huerta, D. Lesny, and M. Neubauer, "Container Solutions for HPC Systems: A Case Study of Using Shifter on Blue Waters," New York, NY, USA, 2018.

[33] C. Witt, M. Bux, W. Gusew, and U. Leser, "Predictive Performance Modeling for Distributed Batch Processing Using Black Box Monitoring and Machine Learning," *Information Systems*, vol. 82, pp. 33–52, 2019.

[34] F. Nadeem and T. Fahringer, “Using Templates to Predict Execution Time of Scientific Workflow Applications in The Grid,” in *Proceedings of The 9th IEEE/ACM International Symposium on Cluster Computing and The Grid*, May 2009, pp. 316–323.

[35] R. F. da Silva, G. Juve, M. Rynge, E. Deelman, and M. Livny, “Online Task Resource Consumption Prediction for Scientific Workflows,” *Parallel Processing Letters*, vol. 25, no. 03, p. 1541003, 2015.

[36] T. P. Pham, J. J. Durillo, and T. Fahringer, “Predicting Workflow Task Execution Time in The Cloud Using A Two-Stage Machine Learning Approach,” *IEEE Transactions on Cloud Computing*, pp. 1–1, 2017.

[37] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. J. Wright, “Performance Analysis of High Performance Computing Applications on The Amazon Web Services Cloud,” in *Proceedings of The 2nd IEEE International Conference on Cloud Computing Technology and Science*, Nov. 2010, pp. 159–168.

[38] M. Mao and M. Humphrey, “A Performance Study on The VM Startup Time in The Cloud,” in *Proceedings of The 5th IEEE International Conference on Cloud Computing*, June 2012, pp. 423–430.

[39] M. Jones, B. Arcand, B. Bergeron, D. Bestor, C. Byun, L. Milechin, V. Gadepally, M. Hubbell, J. Kepner, P. Michaleas, J. Mullen, A. Prout, T. Rosa, S. Samsi, C. Yee, and A. Reuther, “Scalability of VM Provisioning Systems,” in *Proceedings of The IEEE High Performance Extreme Computing Conference*, Sep. 2016, pp. 1–5.

[40] M. A. Murphy, B. Kagey, M. Fenn, and S. Goasguen, “Dynamic Provisioning of Virtual Organization Clusters,” in *Proceedings of The 9th IEEE/ACM International Symposium on Cluster Computing and The Grid*, May 2009, pp. 364–371.

[41] W. Chen, Y. C. Lee, A. Fekete, and A. Y. Zomaya, “Adaptive Multiple-workflow Scheduling with Task Rearrangement,” *The Journal of Supercomputing*, vol. 71, no. 4, pp. 1297–1317, Apr. 2015.

[42] Y. R. Wang, K. C. Huang, and F. J. Wang, “Scheduling Online Mixed-parallel Workflows of Rigid Tasks in Heterogeneous Multi-cluster Environments,” *Future Generation Computer Systems*, vol. 60, pp. 35–47, 2016.

[43] A. Hamid, G. B. Jorge, and S. Frédéric, “Fair Resource Sharing for Dynamic Scheduling of Workflows on Heterogeneous Systems,” in *High-Performance Computing on Complex Environments*. John Wiley & Sons, Ltd, 2014, ch. 9, pp. 145–167.

[44] A. C. Zhou, B. He, and C. Liu, “Monetary Cost Optimizations for Hosting Workflow-as-a-Service in IaaS Clouds,” *IEEE Transactions on Cloud Computing*, vol. 4, no. 1, pp. 34–48, Jan 2016.

[45] Z. Yu and W. Shi, “A Planner-Guided Scheduling Strategy for Multiple Workflow Applications,” in *Proceedings of The International Conference on Parallel Processing*, Sep. 2008, pp. 1–8.

[46] H. Topcuoglu, S. Hariri, and M. Y. Wu, “Performance-effective and Low-complexity Task Scheduling for Heterogeneous Computing,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, March 2002.

[47] U. Hönig and W. Schiffmann, “A Comprehensive Test Bench for The Evaluation of Scheduling Heuristics,” in *Proceedings of The 16th IASTED International Conference on Parallel and Distributed Computing and Systems*, 2004, pp. 437–442.

[48] M. Xu, L. Cui, H. Wang, and Y. Bi, “A Multiple QoS Constrained Scheduling Strategy of Multiple Workflows for Cloud Computing,” in *Proceedings of The IEEE International Symposium on Parallel and Distributed Processing with Applications*, Aug 2009, pp. 629–634.

[49] C. Lizhen, X. Meng, and Y. Bi, “A Scheduling Strategy for Multiple QoS Constrained Grid Workflows,” in *Proceedings of The Joint Conferences on Pervasive Computing*, Dec 2009, pp. 561–566.

[50] J. G. Barbosa and B. Moreira, “Dynamic Scheduling of A Batch of Parallel Task Jobs on Heterogeneous Clusters,” *Parallel Computing*, vol. 37, no. 8, pp. 428–438, 2011, follow-on of ISPDC’2009 and HeteroPar’2009.

[51] J. G. Barbosa, C. Morais, R. Nobrega, and A. Monteiro, “Static Scheduling of Dependent Parallel Tasks on Heterogeneous Clusters,” in *Proceedings of The IEEE International Conference on Cluster Computing*, Sep. 2005, pp. 1–8.

[52] C. C. Hsu, K. C. Huang, and F. J. Wang, “Online Scheduling of Workflow Applications in Grid Environments,” *Future Generation Computer Systems*, vol. 27, no. 6, pp. 860–870, 2011.

[53] M. Quinson, “SimGrid: A Generic Framework for Large-scale Distributed Experiments,” in *Proceedings of The 9th IEEE International Conference on Peer-to-Peer Computing*, Sep. 2009, pp. 95–96.

[54] F. Cappello, F. Desprez, M. Dayde, E. Jeannot, Y. Jégou, S. Lanteri, N. Melab, R. Namyst, P. Primet, O. Richard, E. Caron, J. Leduc, and G. Mornet, “Grid’5000: A Large-scale, Reconfigurable, Controlable and Monitorable Grid Platform,” in *Proceedings of The 6th IEEE/ACM International Workshop on Grid Computing*, Seattle, USA, United States, Nov. 2005, grid 2005 held in conjunction with SC’05, the International Conference for High Performance Computing, Networking and Storage.

[55] H. Arabnejad and J. G. Barbosa, “Maximizing The Completion Rate of Concurrent Scientific Applications Under Time and Budget Constraints,” *Journal of Computational Science*, vol. 23, pp. 120–129, 2017.

[56] ——, “Multi-QoS Constrained and Profit-aware Scheduling Approach for Concurrent Workflows on Heterogeneous Systems,” *Future Generation Computer Systems*, vol. 68, pp. 211–221, 2017.

[57] G. L. Stavrinides and H. D. Karatza, “Scheduling Multiple Task Graphs in Heterogeneous Distributed Real-time Systems by Exploiting Schedule Holes with Bin Packing Techniques,” *Simulation Modelling Practice and Theory*, vol. 19, no. 1, pp. 540–552, 2011, Modeling and Performance Analysis of Networking and Collaborative Systems.

[58] ——, "Scheduling Multiple Task Graphs with End-to-end Deadlines in Distributed Real-time Systems Utilizing Imprecise Computations," *Journal of Systems and Software*, vol. 83, no. 6, pp. 1004–1014, 2010, Software Architecture and Mobility.

[59] ——, "A Cost-effective and QoS-aware Approach to Scheduling Real-time Workflow Applications in PaaS and SaaS Clouds," in *Proceedings of The 3rd International Conference on Future Internet of Things and Cloud*, Aug 2015, pp. 231–239.

[60] ——, "The Impact of Resource Heterogeneity on The Timeliness of Hard Real-time Complex Jobs," in *Proceedings of the 7th International Conference on PErvasive Technologies Related to Assistive Environments*, ser. PETRA '14. New York, NY, USA: Association for Computing Machinery, 2014.

[61] G. L. Stavrinides, F. R. Duro, H. D. Karatza, J. G. Blas, and J. Carretero, "Different Aspects of Workflow Scheduling in Large-scale Distributed Systems," *Simulation Modelling Practice and Theory*, vol. 70, pp. 120–134, 2017.

[62] F. R. Duro, J. G. Blas, and J. Carretero, "A Hierarchical Parallel Storage System Based on Distributed Memory for Large Scale Systems," in *Proceedings of The 20th European MPI Users' Group Meeting*, ser. EuroMPI '13. New York, NY, USA: Association for Computing Machinery, 2013, pp. 139—140.

[63] S. Sharif, J. Taheri, A. Y. Zomaya, and S. Nepal, "Online Multiple Workflow Scheduling under Privacy and Deadline in Hybrid Cloud Environment," in *Proceedings of The 6th IEEE International Conference on Cloud Computing Technology and Science*, Dec 2014, pp. 455–462.

[64] P. Watson, "A Multi-level Security Model for Partitioning Workflows Over Federated Clouds," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 1, no. 1, p. 15, Jul 2012.

[65] Y. Tsai, H. Liu, and K. Huang, "Adaptive Dual-criteria Task Group Allocation for Clustering-based Multi-workflow Scheduling on Parallel Computing Platform," *The Journal of Supercomputing*, vol. 71, no. 10, pp. 3811–3831, Oct. 2015.

[66] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M. H. Su, and K. Vahi, "Characterization of Scientific Workflows," in *Proceedings of The 3rd Workshop on Workflows in Support of Large-Scale Science*, Nov. 2008, pp. 1–10.

[67] B. Lin, W. Guo, and X. Lin, "Online Optimization Scheduling for Scientific Workflows with Deadline Constraint on Hybrid Clouds," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 11, pp. 3079–3095, 2016.

[68] H. Chen, X. Zhu, D. Qiu, and L. Liu, "Uncertainty-aware Real-time Workflow Scheduling in The Cloud," in *Proceedings of The 9th IEEE International Conference on Cloud Computing*, June 2016, pp. 577–584.

[69] X. Tang, K. Li, G. Liao, K. Fang, and F. Wu, "A Stochastic Scheduling Algorithm for Precedence Constrained Tasks on Grid," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1083–1091, 2011.

[70] D. Poola, S. Garg, R. Buyya, Y. Yang, and K. Ramamohanarao, "Robust Scheduling of Scientific Workflows with Deadline and Budget Constraints in Clouds," in *Proceedings of The 28th IEEE International Conference on Advanced Information Networking and Applications*, May 2014, pp. 858–865.

[71] H. Chen, J. Zhu, Z. Zhang, M. Ma, and X. Shen, "Real-time Workflows Oriented Online Scheduling in Uncertain Cloud Environment," *The Journal of Supercomputing*, vol. 73, no. 11, pp. 4906–4922, Nov. 2017.

[72] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[73] H. Chen, X. Zhu, D. Qiu, H. Guo, L. T. Yang, and P. Lu, "EONS: Minimizing Energy Consumption for Executing Real-Time Workflows in Virtualized Cloud Data Centers," in *Proceedings of The 45th International Conference on Parallel Processing Workshops*, Aug 2016, pp. 385–392.

[74] V. Ebrahimirad, M. Goudarzi, and A. Rajabi, “Energy-Aware Scheduling for Precedence-Constrained Parallel Virtual Machines in Virtualized Data Centers,” *Journal of Grid Computing*, vol. 13, no. 2, pp. 233–253, Jun 2015.

[75] I. Pietri and R. Sakellariou, “Energy-Aware Workflow Scheduling Using Frequency Scaling,” in *Proceedings of The 43rd International Conference on Parallel Processing Workshops*, Sep. 2014, pp. 104–113.

[76] X. Zhu, J. Wang, H. Guo, D. Zhu, L. T. Yang, and L. Liu, “Fault-Tolerant Scheduling for Real-Time Scientific Workflows with Elastic Resource Provisioning in Virtualized Clouds,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 12, pp. 3501–3517, Dec 2016.

[77] X. Qin and H. Jiang, “A Novel Fault-tolerant Scheduling Algorithm for Precedence Constrained Tasks in Real-time Heterogeneous Systems,” *Parallel Computing*, vol. 32, no. 5, pp. 331–356, 2006.

[78] H. Chen, X. Zhu, G. Liu, and W. Pedrycz, “Uncertainty-Aware Online Scheduling for Real-Time Workflows in Cloud Service Environment,” *IEEE Transactions on Services Computing*, pp. 1–1, 2018.

[79] H. Chen, J. Zhu, G. Wu, and L. Huo, “Cost-efficient Reactive Scheduling for Real-time Workflows in Clouds,” *The Journal of Supercomputing*, vol. 74, no. 11, pp. 6291–6309, Nov 2018.

[80] M. A. Rodriguez and R. Buyya, “Scheduling Dynamic Workloads in Multi-tenant Scientific Workflow as a Service Platforms,” *Future Generation Computer Systems*, vol. 79, pp. 739–750, 2018.

[81] X. Xu, W. Dou, X. Zhang, and J. Chen, “EnReal: An Energy-aware Resource Allocation Method for Scientific Workflow Executions in Cloud Environment,” *IEEE Transactions on Cloud Computing*, vol. 4, no. 2, pp. 166–179, April 2016.

[82] X. Liu, Y. Yang, Y. Jiang, and J. Chen, "Preventing Temporal Violations in Scientific Workflows: Where and How," *IEEE Transactions on Software Engineering*, vol. 37, no. 6, pp. 805–825, Nov 2011.

[83] S. Zhang, B. Wang, B. Zhao, and J. Tao, "An Energy-Aware Task Scheduling Algorithm for a Heterogeneous Data Center," in *Proceedings of The 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, July 2013, pp. 1471–1477.

[84] J. J. Durillo, H. M. Fard, and R. Prodan, "MOHEFT: A Multi-objective List-based Method for Workflow Scheduling," in *Proceedings of The 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, Dec 2012, pp. 185–192.

[85] Y. Wang, S. Cao, G. Wang, Z. Feng, C. Zhang, and H. Guo, "Fairness Scheduling with Dynamic Priority for Multi Workflow on Heterogeneous Systems," in *Proceedings of The 2nd IEEE International Conference on Cloud Computing and Big Data Analysis*, April 2017, pp. 404–409.

[86] G. Z. Tian, C. B. Xiao, Z. S. Xu, and X. Xiao, "Hybrid Scheduling Strategy for Multiple DAGs Workflow in Heterogeneous System," *Ruanjian Xuebao/Journal of Software*, vol. 23, no. 10, pp. 2720–2734, 2012.

[87] G. Wang, Y. Wang, H. Liu, and H. Guo, "HSIP: A Novel Task Scheduling Algorithm for Heterogeneous Computing," *Sci. Program.*, vol. 2016, p. 19, Mar. 2016.

[88] G. Xie, L. Liu, L. Yang, and R. Li, "Scheduling Trade-off of Dynamic Multiple Parallel Workflows on Heterogeneous Distributed Computing Systems," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 2, p. e3782, 2017, e3782 cpe.3782.

[89] G. Xie, R. Li, X. Xiao, and Y. Chen, "A High-Performance DAG Task Scheduling Algorithm for Heterogeneous Networked Embedded Systems," in *Proceedings of The 28th IEEE International Conference on Advanced Information Networking and Applications*, May 2014, pp. 1011–1016.

[90] G. Xie, G. Zeng, J. Jiang, C. Fan, R. Li, and K. Li, "Energy Management for Multiple Real-time Workflows on Cyber–physical Cloud Systems," *Future Generation Computer Systems*, vol. 105, pp. 916–931, 2020.

[91] Z. Tang, L. Qi, Z. Cheng, K. Li, S. U. Khan, and K. Li, "An Energy-Efficient Task Scheduling Algorithm in DVFS-enabled Cloud Environment," *Journal of Grid Computing*, vol. 14, no. 1, pp. 55–74, Mar 2016.

[92] K. Li, "Scheduling Precedence Constrained Tasks with Reduced Processor Energy on Multiprocessor Computers," *IEEE Transactions on Computers*, vol. 61, no. 12, pp. 1668–1681, Dec 2012.

[93] J. Livny, H. Teonadi, M. Livny, and M. K. Waldor, "High-Throughput, Kingdom-wide Prediction and Annotation of Bacterial Non-coding RNAs," *PLOS ONE*, vol. 3, no. 9, pp. 1–12, 09 2008.

[94] N. Zhou, F. Li, K. Xu, and D. Qi, "Concurrent Workflow Budget- and Deadline-constrained Scheduling in Heterogeneous Distributed Environments," *Soft Computing*, vol. 22, no. 23, pp. 7705–7718, Dec 2018.

[95] J. Liu, J. Ren, W. Dai, D. Zhang, P. Zhou, Y. Zhang, G. Min, and N. Najjari, "Online Multi-Workflow Scheduling under Uncertain Task Execution Time in IaaS Clouds," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2019.

[96] M. Mao and M. Humphrey, "Auto-scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows," in *Proceedings of The IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2011, pp. 1–12.

[97] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Deadline-constrained Workflow Scheduling Algorithms for Infrastructure as a Service Clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, 2013, including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures.

[98] M. Malawski, K. Figiela, M. Bubak, E. Deelman, and J. Nabrzyski, "Scheduling Multi-level Deadline-constrained Scientific Workflows on Clouds based on Cost Optimization," *Scientific Programming*, vol. 2015, Jan. 2015.

[99] V. Arabnejad, K. Bubendorfer, and B. Ng, "Deadline Distribution Strategies for Scientific Workflow Scheduling in Commercial Clouds," in *Proceedings of The ACM/IEEE International Conference on Utility and Cloud Computing*, ser. UCC '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 70–78.

[100] Z. Cai, X. Li, and R. Ruiz, "Resource Provisioning for Task-batch based Workflows with Deadlines in Public Clouds," *IEEE Transactions on Cloud Computing*, vol. 7, no. 3, pp. 814–826, July 2019.

[101] F. Wu, Q. Wu, Y. Tan, R. Li, and W. Wang, "PCP-B^2: Partial Critical Path Budget Balanced Scheduling Algorithms for Scientific Workflow Applications," *Future Generation Computer Systems*, vol. 60, pp. 22–34, 2016.

[102] C. Q. Wu, X. Lin, D. Yu, W. Xu, and L. Li, "End-to-end Delay Minimization for Scientific Workflows in Clouds under Budget Constraint," *IEEE Transactions on Cloud Computing*, vol. 3, no. 2, pp. 169–181, April 2015.

[103] X. Wang, B. Cao, C. Hou, L. Xiong, and J. Fan, "Scheduling Budget Constrained Cloud Workflows with Particle Swarm Optimization," in *Proceedings of The IEEE Conference on Collaboration and Internet Computing*, Oct 2015, pp. 219–226.

[104] A. Verma and S. Kaushal, "Budget Constrained Priority based Genetic Algorithm for Workflow Scheduling in Cloud," in *Proceedings of The International Conference on Advances in Recent Technologies in Communication and Computing*. Institution of Engineering and Technology, January 2013, pp. 216–222.

[105] ——, "Deadline and Budget Distribution based Cost-Time Optimization Workflow Scheduling Algorithm for Cloud," in *Proceedings of The International Conference on Recent Advances and Future Trends in Information Technology*. IJCA, 2012.

[106] M. A. Rodriguez and R. Buyya, "Budget-Driven Resource Provisioning and Scheduling of Scientific Workflow in IaaS Clouds with Fine-grained Billing Periods," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 12, no. 2, May 2017.

[107] V. Arabnejad, K. Bubendorfer, and B. Ng, "Budget Distribution Strategies for Scientific Workflow Scheduling in Commercial Clouds," in *Proceedings of The IEEE International Conference on e-Science*, Oct 2016, pp. 137–146.

[108] J. Yu, R. Buyya, and C. K. Tham, "Cost-based Scheduling of Scientific Workflow Applications on Utility Grids," in *Proceedings of The IEEE International Conference on e-Science and Grid Computing*, July 2005.

[109] Y. Yuan, X. Li, Q. Wang, and Y. Zhang, "Bottom Level based Heuristic for Workflow Scheduling in Grids," *Chinese Journal of Computers*, vol. 31, no. 2, p. 282, 2008.

[110] E. N. Alkhanak, S. P. Lee, R. Rezaei, and R. M. Parizi, "Cost Optimization Approaches for Scientific Workflow Scheduling in Cloud and Grid Computing: A Review, Classifications, and Open Issues," *Journal of Systems and Software*, vol. 113, pp. 1–26, 2016.

[111] R. Qasha, J. Cala, and P. Watson, "Dynamic Deployment of Scientific Workflows in The Cloud Using Container Virtualization," in *Proceedings of The IEEE International Conference on Cloud Computing Technology and Science*, Dec 2016, pp. 269–276.

[112] K. Liu, K. Aida, S. Yokoyama, and Y. Masatani, "Flexible Container-based Computing Platform on Cloud for Scientific Workflows," in *Proceedings of The International Conference on Cloud Computing Research and Innovations*, May 2016, pp. 56–63.

[113] E. J. Alzahrani, Z. Tari, Y. C. Lee, D. Alsadie, and A. Y. Zomaya, "adCFS: Adaptive Completely Fair Scheduling Policy for Containerised Workflows Systems," in *Proceedings of The 16th IEEE International Symposium on Network Computing and Applications*, Oct 2017, pp. 1–8.

[114] H. Cao and C. Q. Wu, "Performance Optimization of Budget-constrained MapReduce Workflows in Multi-clouds," in *Proceedings of The 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, May 2018, pp. 243–252.

[115] Y. Caniou, E. Caron, A. K. W. Chang, and Y. Robert, "Budget-Aware Scheduling Algorithms for Scientific Workflows with Stochastic Task Weights on Heterogeneous IaaS Cloud Platforms," in *Proceedings of The IEEE International Parallel and Distributed Processing Symposium Workshops*, May 2018, pp. 15–26.

[116] V. Arabnejad, K. Bubendorfer, and B. Ng, "A Budget-Aware Algorithm for Scheduling Scientific Workflows in Cloud," in *Proceedings of The 18th IEEE International Conference on High Performance Computing and Communications; 14th IEEE International Conference on Smart City; 2nd IEEE International Conference on Data Science and Systems*, Dec 2016, pp. 1188–1195.

[117] W. Chen, G. Xie, R. Li, Y. Bai, C. Fan, and K. Li, "Efficient Task Scheduling for Budget Constrained Parallel Applications on Heterogeneous Cloud Computing Systems," *Future Generation Computer Systems*, vol. 74, pp. 1–11, 2017.

[118] Z. Kozhirbayev and R. O. Sinnott, "A Performance Comparison of Container-based Technologies for The Cloud," *Future Generation Computer Systems*, vol. 68, pp. 175–182, 2017.

[119] R. Shea, F. Wang, H. Wang, and J. Liu, "A Deep Investigation Into Network Performance in Virtual Machine Based Cloud Environments," in *Proceeding of The IEEE Conference on Computer Communications*, April 2014, pp. 1285–1293.

[120] M. Ullrich, J. Lässig, J. Sun, M. Gaedke, and K. Aida, "A Benchmark Model for The Creation of Compute Instance Performance Footprints," in *Internet and Distributed Computing Systems*. Cham: Springer International Publishing, 2018, pp. 221–234.

[121] S. F. Piraghaj, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, "ContainerCloudsim: An Environment for Modeling and Simulation of Containers in Cloud Data Centers," *Software: Practice and Experience*, vol. 47, no. 4, pp. 505–521, 2017.

[122] G. Kousalya, P. Balakrishnan, and C. Pethuru Raj, *Workflow Scheduling Algorithms and Approaches*. Cham: Springer International Publishing, 2017, pp. 65–83.

[123] F. Nadeem and T. Fahringer, "Optimizing Execution Time Predictions of Scientific Workflow Applications in the Grid Through Evolutionary Programming," *Future Generation*

Computer Systems, vol. 29, no. 4, pp. 926–935, 2013, Special Section: Utility and Cloud Computing.

[124] S. Pumma, W. chun Feng, P. Phunchongharn, S. Chapeland, and T. Achalakul, "A Runtime Estimation Framework for ALICE," *Future Generation Computer Systems*, vol. 72, pp. 65–77, 2017.

[125] A. Matsunaga and J. A. B. Fortes, "On the Use of Machine Learning to Predict the Time and Resources Consumed by Applications," in *Proceedings of The 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, May 2010, pp. 495–504.

[126] D. A. Monge, M. Holec, F. Železný, and C. Garino, "Ensemble Learning of Runtime Prediction Models for Gene-expression Analysis Workflows," *Cluster Computing*, vol. 18, no. 4, pp. 1317–1329, Dec 2015.

[127] S. Seneviratne and D. C. Levy, "Task Profiling Model for Load Profile Prediction," *Future Generation Computer Systems*, vol. 27, no. 3, pp. 245–255, 2011.

[128] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to Forget: Continual Prediction with LSTM," *IET Conference Proceedings*, pp. 850–855, January 1999.

[129] W. Groß, S. Lange, J. Boedecker, and M. Blum, "Predicting Time Series with Space-Time Convolutional and Recurrent Neural Networks," in *Proceeding of European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2017, pp. 71–76.

[130] T. A. Babu and P. R. Kumar, "Characterization and Classification of Uterine Magnetomyography Signals Using KNN Classifier," in *Proceeding of The Conference on Signal Processing And Communication Engineering Systems*, Jan 2018, pp. 163–166.

[131] B. Li, Y. Zhang, M. Jin, T. Huang, and Y. Cai, "Prediction of Protein-Peptide Interactions with a Nearest Neighbor Algorithm," *Current Bioinformatics*, vol. 13, no. 1, pp. 14–24, 2018.

[132] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based Learning Algorithms," *Machine Learning*, vol. 6, no. 1, pp. 37–66, Jan 1991.

[133] E. Frank, M. A. Hall, G. Holmes, R. Kirkby, B. Pfahringer, I. H. Witten, and L. Trigg, *Weka-A Machine Learning Workbench for Data Mining*. Boston, MA: Springer US, 2010, pp. 1269–1277.

[134] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. F. da Silva, M. Livny, and K. Wenger, "Pegasus, A Workflow Management System for Science Automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015.

[135] E. Deelman, C. Carothers, A. Mandal, B. Tierney, J. S. Vetter, I. Baldin, C. Castillo, G. Juve, D. Król, V. Lynch, B. Mayer, J. Meredith, T. Proffen, P. Ruth, and R. F. da Silva, "PANORAMA: An Approach to Performance Modeling and Diagnosis of Extreme-scale Workflows," *The International Journal of High Performance Computing Applications*, vol. 31, no. 1, pp. 4–18, 2017.

[136] M. A. Hall, "Correlation-based Feature Selection for Machine Learning," Ph.D. dissertation, University of Waikato, Hamilton, 1999.

[137] B. D. Fulcher and N. S. Jones, "Highly Comparative Feature-Based Time-Series Classification," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 12, pp. 3026–3037, Dec 2014.

[138] O. Trott and A. J. Olson, "AutoDock Vina: Improving the Speed and Accuracy of Docking with A New Scoring Function, Efficient Optimization, and Multithreading," *Journal of Computational Chemistry*, vol. 31, no. 2, pp. 455–461, 2010.

[139] J. Armstrong and F. Collopy, "Error Measures for Generalizing About Forecasting Methods: Empirical Comparisons," *International Journal of Forecasting*, vol. 8, no. 1, pp. 69–80, 1992.

[140] T. Fahringer, R. Prodan, R. Duan, J. Hofer, F. Nadeem, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H. L. Truong, A. Villazon, and M. Wieczorek, *ASKALON: A Development and Grid Computing Environment for Scientific Workflows*. London: Springer London, 2007, pp. 450–471.

[141] P. Blaha, K. Schwarz, G. K. Madsen, D. Kvasnicka, and J. Luitz, "WIEN2K, An Augmented Plane Wave+ Local Orbitals Program for Calculating Crystal Properties, edited by K," *Schwarz, Vienna University of Technology, Austria*, 2001.

[142] J. Goecks, A. Nekrutenko, J. Taylor, and T. G. Team, "Galaxy: A Comprehensive Approach for Supporting Accessible, Reproducible, and Transparent Computational Research in The Life Sciences," *Genome Biology*, vol. 11, no. 8, p. R86, 2010.

[143] C. A. Goble, J. Bhagat, S. Aleksejevs, D. Cruickshank, D. Michaelides, D. Newman, M. Borkum, S. Bechhofer, M. Roos, P. Li, and D. De Roure, "myExperiment: A Repository and Social Network for The Sharing of Bioinformatics Workflows," *Nucleic Acids Research*, vol. 38, no. suppl_2, pp. 677–682, 05 2010.

[144] D. R. Bharti, A. J. Hemrom, and A. M. Lynn, "GCAC: Galaxy Workflow System for Predictive Model Building for Virtual Screening," *BMC Bioinformatics*, vol. 19, no. 13, p. 550, 2019.

[145] M. W. C. Thang, X. Y. Chua, G. Price, D. Gorse, and M. A. Field, "MetaDEGalaxy: Galaxy Workflow for Differential Abundance Analysis of 16s Metagenomic Data," *F1000Research*, vol. 8, pp. 726–726, May 2019, 31737256[pmid].

[146] D. Eisler, D. Fornika, L. C. Tindale, T. Chan, S. Sabaiduc, R. Hickman, C. Chambers, M. Krajden, D. M. Skowronski, A. Jassem, and W. Hsiao, "Influenza Classification Suite: An Automated Galaxy Workflow for Rapid Influenza Sequence Analysis," *Influenza and Other Respiratory Viruses*, vol. n/a, no. n/a.

[147] M. Malawski, A. Gajek, A. Zima, B. Balis, and K. Figiela, "Serverless Execution of Scientific Workflows: Experiments with HyperFlow, AWS Lambda and Google Cloud Functions," *Future Generation Computer Systems*, 2017.

[148] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock, "Kepler: An Extensible System for Design and Execution of Scientific Workflows," in *Proceedings of The 16th International Conference on Scientific and Statistical Database Management*, June 2004, pp. 423–424.

[149] J. Davis II, M. Goel, C. Hylands, B. Kienhuis, E. A. Lee, J. Liu, X. Liu, L. Muliadi, S. Neuendorffer, J. Reekie *et al.*, "Overview of the Ptolemy Project," ERL Technical Report UCB/ERL, Tech. Rep., 1999.

[150] P. Korambath, J. Wang, A. Kumar, J. Davis, R. Graybill, B. Schott, and M. Baldea, "A Smart Manufacturing Use Case: Furnace Temperature Balancing in Steam Methane Reforming Process via Kepler Workflows," *Procedia Computer Science*, vol. 80, pp. 680–689, 2016, international Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA.

[151] P. C. Yang, S. Purawat, P. U. Ieong, M. T. Jeng, K. R. DeMarco, I. Vorobyov, A. D. McCulloch, I. Altintas, R. E. Amaro, and C. E. Clancy, "A Demonstration of Modularity, Reuse, Reproducibility, Portability and Scalability for Modeling and Simulation of Cardiac Electrophysiology Using Kepler Workflows," *PLOS Computational Biology*, vol. 15, no. 3, pp. 1–19, 03 2019.

[152] D. Thain, T. Tannenbaum, and M. Livny, "Distributed Computing in Practice: The Condor Experience," *Concurrency - Practice and Experience*, vol. 17, no. 2-4, pp. 323–356, 2005.

[153] E. Deelman, C. Kesselman, G. Mehta, L. Meshkat, L. Pearlman, K. Blackburn, P. Ehrens, A. Lazzarini, R. Williams, and S. Koranda, "GriPhyN and LIGO, Building a Virtual Data Grid for Gravitational Wave Scientists," in *High Performance Distributed Computing*, 2002.

[154] B. B. Misra, *Open-Source Software Tools, Databases, and Resources for Single-Cell and Single-Cell-Type Metabolomics*. New York, NY: Springer New York, 2020, pp. 191–217.

[155] R. Tsonaka, M. Signorelli, E. Sabir, A. Seyer, K. Hettne, A. Aartsma-Rus, and P. Spitali, "Longitudinal Metabolomic Analysis of Plasma Enables Modeling Disease Progression in Duchenne Muscular Dystrophy Mouse Models," *Human Molecular Genetics*, 01 2020, ddz309.

[156] J. Yu and R. Buyya, "Gridbus Workflow Enactment Engine," *Grid Computing: Infrastructure, Service, and Applications*, p. 119, 2018.

[157] C. Vecchiola, X. Chu, and R. Buyya, "Aneka: A Software Platform for .NET-based Cloud Computing," *High Speed and Large Scale Scientific Computing*, vol. 18, pp. 267–295, 2009.

[158] S. Pandey, D. Karunamoorthy, and R. Buyya, *Workflow Engine for Clouds*. John Wiley & Sons, Ltd, 2011, ch. 12, pp. 321–344.

[159] M. P. Mackley, B. Fletcher, M. Parker, H. Watkins, and E. Ormondroyd, "Stakeholder Views on Secondary Findings in Whole-genome and Whole-exome Sequencing: A Systematic Review of Quantitative and Qualitative Studies," *Genetics in Medicine*, vol. 19, no. 3, pp. 283–293, 2017.

[160] D. Dong, Z. Xu, W. Zhong, and S. Peng, "Parallelization of Molecular Docking: A Review," *Current Topics in Medicinal Chemistry*, vol. 18, no. 12, pp. 1015–1028, 2018.

[161] J. Kelleher, Y. Wong, A. W. Wohns, C. Fadil, P. K. Albers, and G. McVean, "Inferring Whole-genome Histories in Large Population Datasets," *Nature Genetics*, vol. 51, no. 9, pp. 1330–1338, 2019.

[162] M. H. Hilman, M. A. Rodríguez, and R. Buyya, "Task Runtime Prediction in Scientific Workflows Using an Online Incremental Learning Approach," in *Proceedings of The 11th IEEE/ACM International Conference on Utility and Cloud Computing*, Dec 2018, pp. 93–102.

[163] A. Gimeno, M. J. Ojeda-Montes, S. Tomás-Hernández, A. Cereto-Massagué, R. Beltrán-Debón, M. Mulero, G. Pujadas, and S. Garcia-Vallvé, "The Light and Dark Sides of Virtual Screening: What Is There to Know?" *International Journal of Molecular Sciences*, vol. 20, no. 6, 2019.

[164] C. Grebner, E. Malmerberg, A. Shewmaker, J. Batista, A. Nicholls, and J. Sadowski, "Virtual Screening in the Cloud: How Big Is Big Enough?" *Journal of Chemical Information and Modeling*, Nov 2019.

[165] E. Deelman, T. Peterka, I. Altintas, C. D. Carothers, K. K. van Dam, K. Moreland, M. Parashar, L. Ramakrishnan, M. Taufer, and J. Vetter, "The Future of Scientific Work-

flows," *The International Journal of High Performance Computing Applications*, vol. 32, no. 1, pp. 159–175, 2018.

[166] M. Fazio, A. Celesti, R. Ranjan, C. Liu, L. Chen, and M. Villari, "Open Issues in Scheduling Microservices in The Cloud," *IEEE Cloud Computing*, vol. 3, no. 5, pp. 81–88, Sep. 2016.

[167] W. Gerlach, W. Tang, A. Wilke, D. Olson, and F. Meyer, "Container Orchestration for Scientific Workflows," in *Proceedings of The IEEE International Conference on Cloud Engineering*, March 2015, pp. 377–378.

[168] T. Combe, A. Martin, and R. D. Pietro, "To Docker or Not to Docker: A Security Perspective," *IEEE Cloud Computing*, vol. 3, no. 5, pp. 54–62, Sep. 2016.

[169] G. M. Kurtzer, V. Sochat, and M. W. Bauer, "Singularity: Scientific Containers for Mobility of Compute," *PLOS ONE*, vol. 12, no. 5, pp. 1–20, 05 2017.

[170] E. Le and D. Paz, "Performance Analysis of Applications Using Singularity Container on SDSC Comet," in *Proceedings of The Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*, ser. PEARC17. New York, NY, USA: Association for Computing Machinery, 2017.

[171] H. Suhartanto, A. P. Pasaribu, M. F. Siddiq, M. I. Fadhila, M. H. Hilman, and A. Yanuar, "A Preliminary Study on Shifting from Virtual Machine to Docker Container for Insilico Drug Discovery in the Cloud," *International Journal of Technology*, vol. 8, no. 4, 2017.

[172] M. Malawski, "Towards Serverless Execution of Scientific Workflows-HyperFlow Case Study," in *Proceedings of The Workshop of Workflows in Support of Large-Scale Sciences*, 2016, pp. 25–33.

[173] Q. Jiang, Y. C. Lee, and A. Y. Zomaya, "Serverless Execution of Scientific Workflows," in *Proceedings of The 15th International Conference Service-Oriented Computing*. Cham: Springer International Publishing, 2017, pp. 706–721.

[174] J. Spillner, C. Mateos, and D. A. Monge, "FaaSter, Better, Cheaper: The Prospect of Serverless Scientific Computing and HPC," in *High Performance Computing*, E. Mocskos and S. Nesmachnow, Eds. Cham: Springer International Publishing, 2018, pp. 154–168.

[175] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft, "Unikernels: Library Operating Systems for the Cloud," in *Proceedings of The 18th International Conference on Architectural Support for Programming Languages and Operating Systems*, vol. 41, no. 1. New York, NY, USA: Association for Computing Machinery, Mar. 2013, p. 461–472.

[176] D. Williams, R. Koller, M. Lucina, and N. Prakash, "Unikernels As Processes," in *Proceedings of The ACM Symposium on Cloud Computing*, ser. SoCC '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 199–211.

[177] F. Jrad, J. Tao, and A. Streit, "A Broker-based Framework for Multi-cloud Workflows," in *Proceedings of The International Workshop on Multi-cloud Applications and Federated Clouds*, ser. MultiCloud '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 61–68.

[178] J. D. Montes, M. Zou, R. Singh, S. Tao, and M. Parashar, "Data-Driven Workflows in Multi-cloud Marketplaces," in *Proceedings of The 7th IEEE International Conference on Cloud Computing*, June 2014, pp. 168–175.

[179] Y. Omote, T. Shinagawa, and K. Kato, "Improving Agility and Elasticity in Bare-metal Clouds," in *Proceedings of The 20th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '15. New York, NY, USA: Association for Computing Machinery, 2015, pp. 145–159.

[180] F. Nadeem, D. Alghazzawi, A. Mashat, K. Fakeeh, A. Almalaise, and H. Hagras, "Modeling and Predicting Execution Time of Scientific Workflows in the Grid Using Radial Basis Function Neural Network," *Cluster Computing*, vol. 20, no. 3, pp. 2805–2819, Sep 2017.

[181] D. Sahoo, S. C. H. Hoi, and B. Li, "Large Scale Online Multiple Kernel Regression with Application to Time-Series Prediction," *ACM Transactions on Knowledge Discovery from Data*, vol. 13, no. 1, Jan. 2019.

[182] J. Zenisek, F. Holzinger, and M. Affenzeller, "Machine Learning based Concept Drift Detection for Predictive Maintenance," *Computers & Industrial Engineering*, vol. 137, p. 106031, 2019.

[183] T. Samak, D. Gunter, M. Goode, E. Deelman, G. Juve, G. Mehta, F. Silva, and K. Vahi, "Online Fault and Anomaly Detection for Large-Scale Scientific Workflows," in *Proceedings of The IEEE International Conference on High Performance Computing and Communications*, Sep. 2011, pp. 373–381.

[184] P. Gaikwad, A. Mandal, P. Ruth, G. Juve, D. Król, and E. Deelman, "Anomaly Detection for Scientific Workflow Applications on Networked Clouds," in *Proceedings of The International Conference on High Performance Computing Simulation*, July 2016, pp. 645–652.

[185] M. A. Rodriguez, R. Kotagiri, and R. Buyya, "Detecting Performance Anomalies in Scientific Workflows Using Hierarchical Temporal Memory," *Future Generation Computer Systems*, vol. 88, pp. 624–635, 2018.

[186] H. Arabnejad and J. G. Barbosa, "Multi-workflow QoS-constrained Scheduling for Utility Computing," in *Proceedings of The 18th IEEE International Conference on Computational Science and Engineering*, Oct 2015, pp. 137–144.

[187] M. Ghasemzadeh, H. Arabnejad, and J. G. Barbosa, "Deadline-Budget constrained Scheduling Algorithm for Scientific Workflows in a Cloud Environment," in *Proceedings of The 20th International Conference on Principles of Distributed Systems*, ser. Leibniz International Proceedings in Informatics (LIPIcs), P. Fatourou, E. Jiménez, and F. Pedone, Eds., vol. 70. Dagstuhl, Germany: Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017, pp. 19:1–19:16.

[188] H. M. Fard, R. Prodan, J. J. Durillo, and T. Fahringer, "A Multi-objective Approach for Workflow Scheduling in Heterogeneous Environments," in *Proceedings of The 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, May 2012, pp. 300–309.

[189] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Algorithms for Cost- and Deadline-constrained Provisioning for Scientific Workflow Ensembles in IaaS Clouds," *Future Generation Computer Systems*, vol. 48, pp. 1–18, 2015, Special Section: Business and Industry Specific Cloud.

[190] A. Beloglazov, R. Buyya, Y. C. Lee, and A. Zomaya, “A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems,” ser. Advances in Computers, M. V. Zelkowitz, Ed. Elsevier, 2011, vol. 82, pp. 47–111.

[191] A. N. Toosi, C. Qu, M. D. de Assunção, and R. Buyya, “Renewable-aware Geographical Load Balancing of Web Applications for Sustainable Data Centers,” *Journal of Network and Computer Applications*, vol. 83, pp. 155–168, 2017.

[192] L. Zeng, B. Veeravalli, and X. Li, “SABA: A Security-aware and Budget-aware Workflow Scheduling Strategy in Clouds,” *Journal of Parallel and Distributed Computing*, vol. 75, pp. 141–151, 2015.

[193] F. Zhao, C. Li, and C. Liu, “A Cloud Computing Security Solution based on Fully Homomorphic Encryption,” in *Proceedings of The 16th International Conference on Advanced Communication Technology*, Feb 2014, pp. 485–488.

[194] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions,” *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013, Including Special Sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services & Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond.

[195] C. Doukas and F. Antonelli, “A Full End-to-end Platform as a Service for Smart City Applications,” in *Proceedings of The 10th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, Oct 2014, pp. 181–186.

[196] M. Nardelli, S. Nastic, S. Dustdar, M. Villari, and R. Ranjan, “Osmotic Flow: Osmotic Computing + IoT Workflow,” *IEEE Cloud Computing*, vol. 4, no. 2, pp. 68–75, March 2017.

[197] G. L. Stavrinides and H. D. Karatza, “A Hybrid Approach to Scheduling Real-time IoT Workflows in Fog and Cloud Environments,” *Multimedia Tools and Applications*, vol. 78, no. 17, pp. 24 639–24 655, Sep 2019.