DOI: 10.14750/ME.2024.035

UNIVERSITY OF MISKOLC

FACULTY OF MECHANICAL ENGINEERING AND INFORMATICS

A Simulation Environment for Modelling and Analysis of Scientific Workflows

PhD dissertation

Author:

ALI AL-HABOOBI

MSc in Advanced Computer Science

"József Hatvany" DOCTORAL SCHOOL OF INFORMATION SCIENCE, ENGINEERING AND TECHNOLOGY

Head of Doctoral School: **Prof. Dr. Jenő SZIGETI**

Academic Supervisor: **Prof. Dr. Gabor KECSKEMETI**

Miskolc
2024

DOI: 10.14750/ME.2024.035

Declaration

I hereby declare that this thesis is my own work unless otherwise stated. No part of this thesis has been previously submitted for a degree or any other qualification at University of Miskolc or any other institution.

Ali Al-Haboobi
Miskolc, 2023. July, 01

DOI: 10.14750/ME.2024.035

Acknowledgments

First of all, I would like to thank my Lord (Allah) for everything He has given me. I would like to thank my supervisor for guiding my doctoral studies. I would like to thank Professor Dr László Kovács for supporting me during my doctoral studies. I would like to thank my parents, my wife and my family for their constant love, prayers and support. Last but not least, I would like to thank my colleagues and friends who helped me to realise the results presented here and to enjoy the time of my studies.

DOI: 10.14750/ME.2024.035

Contents

DOI: 10.14750/ME.2024.035

DOI: 10.14750/ME.2024.035

List of Figures

DOI: 10.14750/ME.2024.035

DOI: 10.14750/ME.2024.035

List of Tables

DOI: 10.14750/ME.2024.035

Chapter 1

Introduction

From the field of manufacturing and business processes, the workflow [60] has evolved into a broader concept that points to a structured design of process flows. The complexity of task execution can vary from sequential execution to highly parallel execution with many inputs from different tasks. Workflows [65] are commonly used in several scientific fields, such as Montage [33] in astronomy, CyberShake [30] in physics and LIGO [1] in astrophysics, to describe complex computational problems and capture data between them. In the scientific community, a scientific workflow consists of many dependent tasks with complex precedence constraints between them. The scientific workflow consists of hundreds or thousands of interdependent computational tasks.

Scientific workflows can be run on distributed computing platforms such as High-Performance Computing (HPC)[32, 66], Grid[7, 57] and Cloud[52]. These platforms offer significant advantages in terms of computing power and scalability, making them ideal for running large-scale scientific applications. However, running workflows on such platforms can be complex and challenging. Workflow management systems (WMS) aim at answering these challenges. WMSs such as Pegasus[22], Kepler[6], and DEWE v3[35] provide a way to manage and handle the execution of workflows through resource selection, job scheduling, appropriate resource allocation and data management. Overall, WMS can significantly improve the efficiency and effectiveness of workflow execution on distributed computing platforms and allow researchers to focus on their scientific goals rather than the technical details of managing and executing their workflows.

Cloud computing is an evolving approach to computing that allows users to access resources based on a usage-based payment model, with the system dynamically adapting to different workload demands. Cloud computing can play an important role in addressing the challenges of scientific workflow applications due to its scalability, reliability and cost-effectiveness.

Scientific workflows have been an increasingly important research area of dis-

DOI: 10.14750/ME.2024.035

tributed systems (such as cloud computing). Researchers have shown an increased interest in the automated processing of scientific applications such as workflows. Function as a Service (FaaS) has recently emerged as a novel distributed systems platform for processing non-interactive applications. FaaS has limitations in resource use (e.g., CPU and RAM) and state management. Despite these, several studies [35, 41, 46] have already demonstrated using FaaS for processing scientific workflows. DEWE v3 [35] can process scientific workflows using AWS Lambda and Google Cloud Functions (GCF). DEWE v3 has three different execution modes: a traditional cluster, a FaaS (serverless), and a hybrid mode (combining the previous two modes).

Conducting real-world experiments for large-scale workflows is challenging. Especially when a statistically significant number of experimental results are required to inform us about possible WMS improvements, this limits the scope of WMS research and development. Therefore, researchers can run a relatively small number of scenarios to substantiate research with real measurements. Moreover, it is very expensive to reproduce experimental results in different real-world scenarios due to resource costs. Therefore, researchers often turn to simulations. The use of computing simulations has become widespread in developing novel techniques, conducting comparative analyses, and understanding and improving the performance of workflow management systems.

Workflow scheduling is an important area for WMS. It plays a critical role in the optimal allocation of resources to all tasks. The problem of scheduling in distributed environments is known to be NP-hard [69]. Therefore, no algorithm can achieve an optimal solution in polynomial time, while some algorithms can give approximate results in polynomial time. When scheduling scientific workflows in the cloud, the deadline constraint refers to the time frame set by the user within which each task must be completed. The scheduling algorithm must consider these deadlines and guarantee that the workflow will be executed within the specified time constraints. Failure to meet these deadlines may result in the workflow being considered failed or incomplete, impacting scheduling algorithms that do not meet the user's deadline.

1.1 Problem Statement

A simulation is an alternative approach to a real experiment that can help evaluate the performance of workflow management systems (WMS) and optimise workflow management techniques. Although several workflow simulators are available today, they [13, 52, 55] are often user-oriented and treat the cloud as a black box. Other workflow simulators [14, 31, 68] cannot meet the requirements of workflow management systems. These requirements include information on virtual machine creation, placement policies, and physical machine schedulers. Unfortunately, this behaviour prevents evaluating the infrastructure-level impact of the various decisions made by

DOI: 10.14750/ME.2024.035

WMSs. In contrast to the above problems, DISSECT-CF [39] is a cloud simulator that captures the internal details of cloud infrastructures. It can be used to develop a more informed WMS simulation. It also provides information on virtual machine creation, placement, and physical machine schedulers. However, DISSECT-CF alone does not provide workflow support.

Function as a Service (FaaS) has recently emerged as a novel distributed systems platform for processing non-interactive applications. FaaS has limitations in resource use (e.g., CPU and RAM) and state management. Despite these, several studies [35, 41, 46] have already demonstrated using FaaS for processing scientific workflows. The workflow management system DEWE v3 executes scientific workflows using FaaS but often suffers from duplicate data transfers while using FaaS. This behaviour is due to handling intermediate data dependency files after and before each function invocation. These data files could fill the temporary storage of the function environment.

Although cloud computing resources can help scientific workflow applications, the problem is finding scheduling algorithms that can optimise the execution of workflows. In the cloud, the cost of executing such workflows depends not only on the number of virtual machines (VMs) but also on the type of these VMs [62]. Selecting the appropriate type and the exact number of VMs is a major challenge for researchers, as tasks in workflow applications are distributed very differently [37]. Algorithms must decide when to provision or de-provision VMs depending on workflow requirements without violating the user's deadline.

1.2 Aims of Research

I. The initial research aim is to introduce an infrastructure simulation for WMS research.

   (a) It should enable researchers to study the internal details of cloud infrastructures and the impact of WMS decisions.

   (b) It should reproduce real-world experiments' behaviours and show our simulation's validity.

   (c) It should simulate serverless execution for scientific workflows based on the behaviour of real-world experiments of Amazon Lambda on DEWE v3.

II. The second research aims to develop a new workflow scheduling algorithm to predict workflow demands in terms of VMs to meet deadlines.

   (a) It should be able to determine the number of VMs and their appropriate type to meet the user's deadline.

DOI: 10.14750/ME.2024.035

(b) It should be able to determine when VMs need to be added or removed to meet the user's deadline.

(c) It should be able to reduce the workflow execution cost without violating the user's deadline.

1.3 Dissertation Guide

Below is the arrangement of the chapters that comprise this dissertation.

Chapter 2 provides the relevant background information and the essential concepts and definitions to fulfil our research aims. It presents the characteristics and structures of scientific workflows. It also contains information about the workflow management systems used in our dissertation. It explains how scientific workflows are scheduled on resources using three scheduling algorithms. Finally, it gives an overview of the cloud environment that is being modelled, an overview of Function as a Service (FaaS), and information about the cloud and workflow simulators used in our dissertation.

Chapter 3 provides the related works of current cloud simulators. It then presents the design and implementation of the DISSECT-CF-WMS simulator. It next provides the performance evaluation of DISSECT-CF-WMS. Finally, we conclude the chapter.

Chapter 4 provides the related works of several deadline-scheduling algorithms for scientific workflows. It then introduces the design and implementation of a Deadline and Structure-Aware Workflow Scheduler (DSAWS). It next provides the performance evaluation of DSAWS with two competitive algorithms. Finally, we conclude the chapter.

Chapter 5 concludes the dissertation, discusses future work and highlights the research efforts of this dissertation.

DOI: 10.14750/ME.2024.035

Chapter 2

Research Background

2.1 Introduction

This chapter explores the relevant background information of this dissertation. The purpose of this chapter is to provide the essential concepts and definitions required to fulfil the aims of our research. It gives an overview of scientific workflows, which are the main topic of this dissertation. We provide background on the structure and characteristics of scientific workflows and the simulator used in the evaluation.

The chapter is structured as follows: Section 2.2 provides a detailed background to scientific workflows, presenting the characteristics and structures of real scientific workflows. Section 2.3 provides information about the workflow management systems used in our dissertation. Section 2.4 explains how scientific workflows are scheduled on resources using three scheduling algorithms. Section 2.5 provides an overview of the cloud environment that is being modelled. Section 2.6 provides an overview of Function as a Service (FaaS). Section 2.7 provides information about the cloud and workflow simulators used in our dissertation. Section 2.8 concludes the chapter.

2.2 Scientific Workflow Representations

A workflow can be represented as a directed acyclic graph (DAG) $G = (T, E)$ consisting of a collection of connected tasks/jobs. Each task/job represents a single work step consisting of a logical step in the overall process. As shown in Figure 2.1, the vertices of the workflow are a set of tasks T=$\{t_1, t_2, ..., t_n\}$, while the workflow edges E represent data dependencies between these tasks [67]. For example, during the execution of the workflow, the successor task t_4 waits for its predecessor task t_1 to complete its processing and produce its output data. When t_1 finishes, some of its output data become input data dependencies for t_4. When t_4 is scheduled, its input

DOI: 10.14750/ME.2024.035

data dependencies are sent to its target host to enable the successful execution of t_4.

Scientific workflows have been widely implemented to allow scientists and engineers to implement more complex applications for accessing and processing large data repositories and running scientific experiments on the Grid or Cloud [64]. However, workflows can be executed in a different environment, where they behave somewhat differently when the same software (applications) runs in a different environment. Therefore, many efforts have been devoted to developing scientific workflow tools that discover their behaviour and information about the current state of the workflow running.

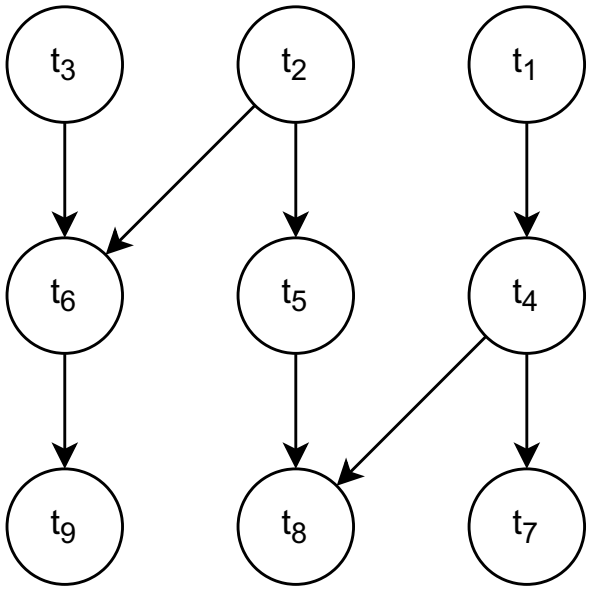


Figure 2.1: *A sample workflow.*

The term dependency in scientific workflows usually refers to a data transfer, such as moving a file. As a result, the five operations that can be depicted in scientific workflows are shown in Figure 2.2: (1) Processing involves one incoming and one outgoing dependency. (2) Pipeline involves each job using the output generated data by the predecessor job, and the output generated serves as input to the successor job for the subsequent stage. (3) Data distribution involves one incoming and multiple outgoing dependencies. (4) Data aggregation involves multiple incoming and one outgoing dependency. (5) Data redistribution involves multiple incoming and outgoing dependencies.

Scientific Workflows [65] are commonly used in several scientific fields, such as Montage [33], CyberShake [30] and LIGO [1], to describe complex computational problems and capture data between them. Workflows can be run on distributed computing platforms such as High-Performance Computing (HPC)[32, 66], Grid[7, 57] and Cloud[52]. These platforms offer significant computing power and scalability

DOI: 10.14750/ME.2024.035

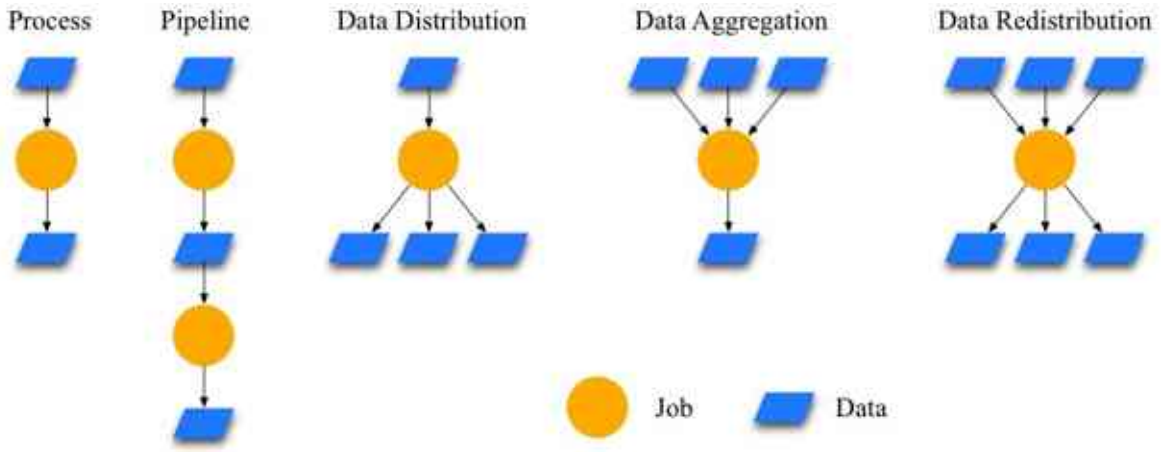


Figure 2.2: *Basic workflow structures [8].*

advantages, making them ideal for running large-scale scientific applications.

In this dissertation, we used a library of realistic workflows introduced by Bharathi et al. [8] to evaluate our research work. All experiments will be evaluated with synthetic/real workflows derived from the applications Montage (astronomy), CyberShake (earthquake science), LIGO (gravitational physics), Epigenomics (bioinformatics), and SIPHT (biology), taking into account data transfers. We used these workflows because they have different structures based on real scientific workflow applications. The Montage[22] workflow is an astronomical application used to generate custom mosaics of the sky based on a set of input images. The CyberShake [30] workflow is used to characterise earthquake hazards by generating synthetic seismograms. The Laser Interferometer Gravitational-Wave Observatory (LIGO) [11] workflow is used to analyse data on the coalescing of compact binary systems such as binary neutron stars and black holes. The Epigenomics workflow represents a largely pipelined application with multiple pipelines operating on distinct chunks of data. The sRNA Identification Protocol using High-throughput Technology (SIPHT) programme [45] uses a workflow to automate the search for sRNA encoding- genes for all bacterial replicons in the National Center for Biotechnology Information (NCBI) database. The structure of the five workflow applications is shown in Figure 2.3.

Over the past decade, the scientific research community has extensively used scientific workflows to take advantage of coarse-grained parallelism in applications running on distributed infrastructures such as clusters[5], grids[7, 57], and clouds [65]. However, these infrastructures have become increasingly complex, diverse, and prone to failure. In addition, scientific workflows have grown in terms of the volume and complexity of data and computations. As a result, numerous techniques, heuristics, and mechanisms have been developed to overcome these challenges and optimise workflow execution.

DOI: 10.14750/ME.2024.035

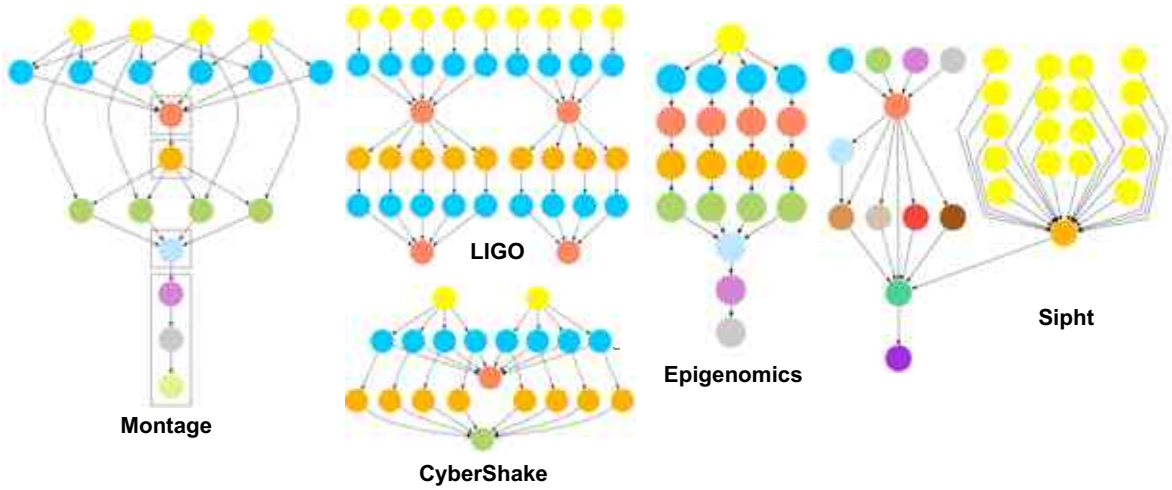


Figure 2.3: *The structure of the Montage, CyberShake, LIGO, Epigenomics and Sipht workflows [8].*

2.3 Workflow Management Systems

Workflow management systems (WMS) are software tools or platforms that enable the design, execution and monitoring of workflows or business processes. They provide a way to streamline and automate the coordination of tasks and data between an organisation's people, departments and systems. WMS typically involves modelling a workflow and managing the execution of those tasks. WMS aims to increase efficiency, reduce errors and improve the quality of results by automating manual processes and providing real-time visibility into the status of work [23].

WMSs such as Pegasus[22], Kepler[6], and DEWE v3[35] provide a way to manage and handle the execution of workflows through resource selection, job scheduling, appropriate resource allocation and data management. Overall, WMS can significantly improve the efficiency and effectiveness of workflow execution on distributed computing platforms and allow researchers to focus on their scientific goals rather than the technical details of managing and executing their workflows.

Several WMSs are being utilised for research and real-life and industry developments. However, we will discuss only Pegasus and DEWE v3, which are directly related to our research. We will highlight them because we used them in our dissertation.

One of the earliest workflow management systems is Pegasus [20], which was initially proposed to execute workflows in grid environments. However, the foundation of Pegasus was laid in 2001, and it is clearly seen in the literature that it is still an active research area thanks to the unfolding of different computing paradigms, such as Cloud Computing and Fog Computing. To execute various workflows with

DOI: 10.14750/ME.2024.035

Pegasus, they offer a description format called DAX (Directed Acyclic Graph in XML).

DEWE v3 can run scientific workflows in three different ways (traditional clusters (VMs), cloud functions, and a hybrid mode that combines both). The FaaS platform supports AWS Lambda and Google Cloud Functions. It has large-scale workflows running on a hybrid approach that combines traditional clusters with the FaaS platform. DEWE v3 runs a workflow engine on a virtual machine. Using AWS Lambda, DEWE v3 reads the workflow definition from an XML file and loads the job binaries and input files into the Amazon S3 object storage. Since Lambda has a temporary storage limit of 500 MB in the execution environment, some jobs cannot be sent to Lambda due to the size of their data files. Jobs that are ready for execution (i.e. according to their precedence constraints) are scheduled into Amazon Kinesis Shards. Each shard acts as an independent queue that can send tasks to its own function instance. The Kinesis batch size determines the number of tasks a function can process in a single invocation. This can be configured before the workflow is executed. Then, the Lambda function pulls a batch of tasks from its own shard to execute them in sequence in a single function invocation. The number of running function instances and the associated kinesis shards can also be configured before the workflow is run and directly influences the maximum degree of parallelism that the execution of the workflow can have.

2.4 Workflow scheduling

Workflow scheduling is an increasingly important area for WMS. It plays a critical role in the optimal allocation of resources to all tasks. The problem of scheduling in distributed environments is known to be NP-hard [69]. Therefore, no algorithm can achieve an optimal solution in polynomial time, while some algorithms can give approximate results in polynomial time. There are several known algorithms for scheduling; we will use the following three in the rest of the dissertation because researchers have widely used them.

MaxMin[10] is a three-stage heuristic algorithm for scheduling. First, it filters all ready tasks (i.e., those for which all input dependencies are met). Then, it sorts the filtered tasks in ascending order according to their expected running time length. Finally, it schedules the task with the longest expected runtime on the best available resource. Consequently, it favours tasks with long runtimes over those with short runtimes.

MinMin[9] is a very similar heuristic algorithm to MaxMin, the difference being mainly in the sort order: unlike MaxMin, MinMin sorts tasks in descending order (again, by their expected runtime). The task with the shortest expected

DOI: 10.14750/ME.2024.035

runtime is then selected to run again on the best available resource. This heuristic aims to create an optimal local path to reduce the total execution time.

Heterogeneous Earliest Finish Time algorithm - HEFT [67] - calculates the average expected execution time of each task on the resources (VMs) and the average communication time of two tasks between all resources. In the first phase, it uses a ranking function to rank the tasks based on the sum of the average execution time and communication time. In the second phase, it assigns a task with the highest ranking value (highest priority) to a resource that would result in minimum execution time.

To execute complex workflows effectively in terms of application makespan, utilisation cost, network utilisation and energy consumption, workflow scheduling algorithms are applied. However, different scenarios may require different scheduling policies. Finding the optimal scheduling for cloud/fog resources is challenging because (i) the availability of resources cannot be foreseen at the time when a task is ready to be executed, (ii) the best fitting resource is hard to find for a task in the case of its migration, and (iii) the maximal utilisation of the resource-constrained fog environment is difficult to manage to achieve the shortest makespan of the workflow application. Finally, (iv) the execution of the workflow has to meet multi-objective criteria [34].

Many scheduling algorithms used in cloud computing systems focus on reducing the execution time of workflow applications, often neglecting other factors such as monetary costs (budgets) or deadlines. Scheduling algorithms aim to establish an appropriate mapping between the tasks of a workflow and the available resources to achieve the application's objective function. This objective function may include or be constrained by optimising a single or multiple Quality of Service (QoS) parameters. The workflow deadline refers to the latest possible time to complete the last task. While a budget is a maximum amount, a user is willing to spend to run a workflow application using computing resources.

2.5 Cloud Computing

Cloud computing [27] refers to the delivery of computing services over the internet, allowing users to access and utilize various resources, such as virtual machines, storage, databases, and software applications, without the need for local infrastructure and hardware. Cloud computing offers several advantages, including cost savings, scalability, and reduced operational overhead. It enables businesses and individuals to focus on their core activities without managing complex infrastructure.

There are many cloud computing service models. We will focus exclusively on the Infrastructure as a Service (IaaS) model, one of the most important service models

DOI: 10.14750/ME.2024.035

in cloud computing. IaaS refers to a cloud computing model in which virtualised computing resources are delivered over the internet. In an IaaS configuration, users can access and manage virtual machines, storage, network infrastructure, and other basic computing resources without investing in or maintaining physical hardware. This allows companies to scale their IT infrastructure dynamically, pay for resources on demand, and focus on their core business without worrying about managing the underlying infrastructure.

Cloud computing is increasingly becoming an important tool for executing workflows [65]. Running workflows on Infrastructure as a Service (IaaS) leads to the challenge of determining the number of virtual machines (VMs) to back the jobs of the workflow. At each workflow stage, there are a different number of jobs, all of which may require different computing resources. Static provisioning (i.e., where a fixed set of VMs are provided for the execution of the workflow from the beginning of its execution to the end) may reduce resource waste and financial expenditure, but it can not improve the performance of the workflow. WMSs must manage the available infrastructure and decide when and how to allocate the resources needed to execute a workflow and how to use them effectively. This requires dynamic provisioning approaches (such as Amazon Auto Scaling [1]) to dynamically add or remove resources based on the workload of the workflow's stages.

2.6 Function as a Service

Function as a Service (FaaS) is a cloud computing model that allows developers to deploy and run code as individual functions. FaaS is a recent development in the field of cloud computing. FaaS is a commercial cloud platform running distributed applications with highly scalable processing capabilities. It promises a simple function-oriented execution environment for non-interactive web application tasks. Like other cloud computing technologies, commercial platforms (such as Amazon Web Services (AWS) Lambda and Google Cloud Functions) are designed to provide FaaS functionalities. These allow functions to run in environments with some limitations. While FaaS offers several benefits, such as scalability, cost-effectiveness, and reduced operational overhead, it also has some limitations. Here are a few limitations of FaaS:

1. Execution Time Limits: FaaS platforms often impose execution time limits on functions. For example, AWS Lambda has a default limit of 15 minutes. Long-running tasks or processes that exceed this limit may not be suitable for FaaS and may require a different approach.

2. Stateless Nature: FaaS functions are stateless, meaning they do not maintain any internal state or memory between invocations. Each function invocation

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

DOI: 10.14750/ME.2024.035

is independent of others. While this design promotes scalability and parallel processing, it can limit applications that rely on maintaining a state or require long-lived connections.

3. Cold Start Overhead: FaaS platforms may need to provision a new container or compute resource to run the function when a function is invoked. This process, known as a cold start, can introduce latency and overhead. Cold starts can be problematic for applications with strict latency requirements or experience sporadic, infrequent invocations.

4. Resource Limitations: FaaS platforms often impose resource limits on functions, such as maximum memory allocation, CPU usage, and disk space. These limits can impact the performance and capabilities of applications that require significant computational resources or have specific hardware requirements.

Considering these limitations when designing and developing applications using FaaS is important. While FaaS can be a powerful tool for certain use cases, it may not be suitable for all applications and workloads.

AWS introduced Lambda[2] in 2014, while Google introduced Cloud Functions (GCF[3]) in 2016. The advantage of using cloud functions is that it dynamically automates resource provisioning by scaling up or down depending on workflow execution requirements. In addition, the billing interval for cloud functions is based on 100 ms, whereas Google Compute Engine and Microsoft Azure recently changed the billing interval for virtual machines from by the hour to by the minute. The function is stateless, and its runtime environment is instantiated and terminated each time the function invokes. In addition, Microsoft and IBM have introduced their own versions of FaaS, namely Microsoft Azure Functions[4] and IBM OpenWhisk Functions[5].

When workflows are run on one of the above FaaS systems, dynamic management of backing VMs by WMS becomes unnecessary, as FaaS systems include automatic resource management in the background. Therefore, the number of concurrent invocations into the infrastructure can better adapt to the actual workflow.

Four FaaS providers, such as Lambda, GCF, Microsoft Azure Functions and OpenWhisk, were evaluated in [26, 43]. The authors proposed multiple hypotheses concerning the expected performance of cloud functions and designed several benchmarks to confirm them. Their function platforms have been tested by invoking CPU, memory, and disk-intensive functions. In addition, data transfer times were also measured for these function providers. They observed different resource allocation policies at the providers. The execution performance of Lambda and GCF is based

[2] https://aws.amazon.com/lambda/
[3] https://cloud.google.com/functions/
[4] https://docs.microsoft.com/en-us/azure/azure-functions/functions-overview
[5] https://cloud.ibm.com/docs/openwhisk?topic=openwhisk-getting-started

DOI: 10.14750/ME.2024.035

on the size of memory allocated for the invocation. They identified that Amazon was more flexible and performant at the time of writing. Moreover, they also reported that computing with cloud functions is more cost-effective than virtual machines due to practically zero delay in booting up new resources. They also indicated that virtual machines would have to sit idle in between invocations due to the more fine-grained invocation patterns to functions. This behaviour results in more costs incurred by virtual machine-based function-oriented solutions. Consequently, we expect more users would prefer Lambda-based workflows due to their efficiency and effectiveness compared with other platforms.

FaaS imposes limitations on the execution of scientific workflows. Since FaaS is designed to execute short-lived, event-driven functions, it may not be suitable for running complex, long-running scientific workflows. The inherent nature of FaaS platforms, where specific events or requests trigger functions, makes it challenging to handle scientific workflows that involve multiple interconnected tasks and potentially long durations. Scientific workflows often require orchestration, coordination, and data dependencies between various computational steps, which may not align well with the stateless and ephemeral nature of FaaS. Moreover, FaaS platforms typically enforce resource limitations, such as execution time limits and memory constraints, to ensure scalability and cost-effectiveness. These limitations can further hinder the execution of scientific workflows that demand significant computational resources or extended processing times.

2.7 Cloud and Workflow Simulators

A simulation-based approach is of great interest in the field of scientific workflow research. DIScrete event-based Energy Consumption simulaTor for Clouds and Federations (DISSECT-CF) has been successfully used to simulate the internals of cloud infrastructures. Figure 2.4 shows the architecture of the currently available [6] 0.9.6 version. The figure groups the main components into subsystems, indicated by dashed lines. Each subsystem is implemented as independently from the others as possible. To perform such simulations, DISSECT-CF has five major subsystems, each responsible for a particular aspect of internal IaaS functionality: (i) event system - for a unified time reference; (ii) unified resource sharing - for solving low-level bottleneck situations; (iii) energy modelling - for analysing the energy utilisation patterns of individual resources (e.g.., network connections, CPUs) or their aggregations; (iv) infrastructure simulation - for modelling PMs, VMs and networked entities; and (v) infrastructure management - to provide infrastructure management.

Using these subsystems, simulations can estimate energy consumption, network

[6] `https://github.com/kecskemeti/dissect-cf`

DOI: 10.14750/ME.2024.035

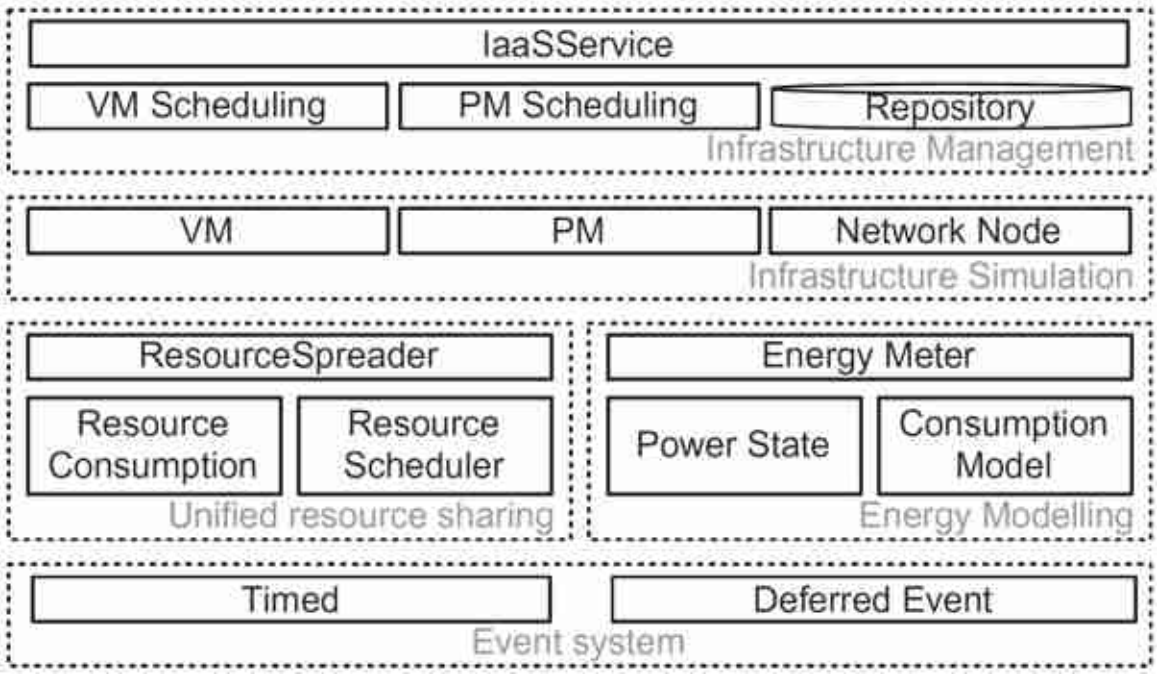


Figure 2.4: *Architectural view of DISSECT-CF [39].*

behaviour, and the impact of virtual machine sharing CPU in various scenarios. DISSECT-CF has shown promising performance gains over popular simulators (e.g. CloudSim, SimGrid). Finally, and most importantly for our workflow simulation goals, DISSECT-CF also provides a simplified network model that allows the modelling of data transfers between workflow tasks. As a result, scheduling techniques based on DISSECT-CF can lead to improved workflow execution times.

This type of information is, at best, only partially available in current commercial and academic cloud (ware) offerings - e.g. Amazon EC2[7], OpenNebula [50] - but DISSECT-CF enables the analysis of user-side schedulers from new perspectives. Based on the results of this analysis, IaaS providers will be able to provide the most useful information to such schedulers in the future.

To optimally support workflows, cloud systems are often used in the background. To increase cost and energy efficiency, workflow systems could build on auto-scaling mechanisms integrated into the clouds. DISSECT-CF's ecosystem also offers several auto-scaling mechanisms that aim to meet the application's requirements running on the infrastructure. In our case, the application would be either a single workflow instance or all workflow instances managed by a particular WMS. Since the simulator's auto-scaling mechanisms are essential for the modern simulation of workflow systems, in the next paragraph, we give an overview of the existing approaches pro-

[7] `https://aws.amazon.com/ec2`

DOI: 10.14750/ME.2024.035

vided by the DISSECT-CF-examples project[8].

The existing auto-scaling mechanisms consider the possible changes to the virtual infrastructure hosted in the cloud every two simulated minutes. Here, we discuss the way the changes are decided. The system automatically collects usage metrics (e.g., CPU usage percentage) for all VMs in the virtual infrastructure. Three different auto-scaling mechanisms currently use this data. First, the ThresholdBasedVI mechanism destroys underutilised VMs and creates a new one only when all other VMs are heavily utilised. The mechanism user can set the threshold that determines which VM is underutilised. Secondly, the VMCreationPriorityBasedVI mechanism applies the same thresholds but favours the creation of VMs over their destruction. Finally, the PoolingVI mechanism keeps some unused VMs in the virtual infrastructure. So, unlike the previous approaches that have to queue tasks, the Pooling approach can accept new jobs at any time during the simulation (since it always has some free virtual machines to which it can direct new jobs). As soon as no more tasks need to be executed by the WMS, the virtual infrastructure is completely dismantled.

WorkflowSim[19], an extension of the widely used CloudSim[13] simulator, is designed to simulate cloud infrastructures and services using the Java programming language. While CloudSim does not support workflow-based simulation by default, WorkflowSim offers several workflow management modules similar to those of Pegasus WMS. These modules include the workflow mapper, which links abstract workflows to concrete jobs in the execution environment. The workflow engine manages dependencies between workflow jobs and the workflow scheduler, which allocates jobs to computing resources. In addition, this tool uses a clustering engine to manage scheduling overhead by merging tasks. However, since WorkflowSim can only run Pegasus trace files via importing DAX files, it is limited to running scientific workflows on cloud resources.

2.8 Summary

In this chapter, we discussed the research background related to our dissertation. In the background, we have discussed concepts that are applied in the later chapters of our dissertation. The chapter provided background information on the structure and characteristics of scientific workflows and the simulator used for the evaluation. This included concepts related to running scientific workflows on a Function as a Service (FaaS) platform, scientific workflow simulators, and designing and implementing a workflow scheduling algorithm.

[8] `https://github.com/kecskemeti/dissect-cf-examples`

DOI: 10.14750/ME.2024.035

Chapter 3

Simulation-based analysis of Internal IaaS behavioural knowledge for a Workflow Management System

3.1 Introduction

In the previous chapter, we improved the scheduling algorithm of DEWE v3 on the FaaS platform (Lambda). We discussed the advantages of running these workflows on FaaS platforms, i.e., Lambda. We have also highlighted the limitations of running scientific workflows on the Lambda platform. Conducting real-world experiments to improve the behaviour of WMSs is a challenge when running large-scale workflows. Especially when a statistically significant number of experimental results are required to inform us about possible WMS improvements, this limits the scope of WMS research and development. Therefore, researchers can run a relatively small number of scenarios to substantiate research with real measurements. Moreover, it is very expensive to reproduce experimental results in different real-world scenarios due to resource costs. Therefore, researchers often turn to simulations.

In this chapter, we focus on the simulation of cloud computing, which has become an important platform for executing workflows because it provides the ability to rent resources on demand in a simple way. Simulation is an emerging area for cloud computing that allows evaluating system performance and improving the behaviour of cloud-based applications. Workflow management systems also use simulation to optimise workflow techniques (e.g., scheduling algorithms). Scientific workflow applications can be evaluated in a simulation environment, resulting in a repeatable and controlled environment. Although the accuracy and validity of simulated results always require final validation, in reality, simulation offers many advantages, such as reproducibility of results, cost efficiency and flexibility. Although many workflow simulators [14, 31, 68] are available today, they cannot meet the requirements of

DOI: 10.14750/ME.2024.035

workflow management systems. These requirements include information about the virtual machine's creation, placement, and scheduling of the physical machine's state. Other cloud simulators [13, 52, 55] are often user-focused and treat clouds as black boxes. Unfortunately, this behaviour prevents the study of the impact of the various decisions made by WMSs at the infrastructure level. Even if a simulator provides insight into the internal workings of clouds, it focuses on some areas (such as precise CPU or network sharing and energy modelling) and ignores others. They thus limit the use cases for these simulators in cloud-aware WMSs [69]. Many workflow simulators [12, 19, 28, 54] do not consider the provisioning delay of a VM in the cloud. This can significantly impact simulation results, especially with auto-scaling that needs to provision and de-provision VMs while a workflow runs in the infrastructure. In contrast to the above problems, DISSECT-CF [39] captures the internal details of cloud infrastructures, which can be used to develop a more informed WMS simulation. It also provides information about virtual machine creation, placement, and schedulers for the physical machine. However, DISSECT-CF alone does not provide workflow support.

Recently, FaaS has been used to run computationally and data-intensive scientific workflows. It provides several desirable features, including (a) automatic provisioning of resources (including CPU, memory, network and temporary storage), (b) automatic scaling as the number of function executions fluctuates over time, and (c) fine-grained pricing model (100 ms). Scientific workflows use these features to solve the problem of over-provisioning, which drives up costs while under-provisioning leads to violation of service level agreements (SLA) and poor quality of service (QoS). However, the use of computing simulations has become widespread in developing novel techniques, conducting comparative analyses, and understanding and improving the performance of workflow management systems.

To address this gap, in this chapter, we present DISSECT-CF-WMS[1], which is built on DISSECT-CF. It was developed to run scientific workflow simulations and investigate internal IaaS behavioural knowledge. First, DISSECT-CF-WMS enables the evaluation of the impact of three physical machine schedulers of a given infrastructure on energy consumption. In addition, it enables better energy awareness by exposing the choice of physical machine schedulers. Second, it can also perform large-scale workflows with good execution simulation performance. Thirdly, DISSECT-CF-WMS has been integrated with the auto-scaling mechanisms of DISSECT-CF to execute scientific workflows allowing WMSs to consider the provisioning delay of a VM in the cloud. Finally, DISSECT-CF-WMS provides a Function as a Service (FaaS) simulation model for running scientific workflows on serverless simulation.

We evaluated our extension by running different workflow applications with the three pre-existing physical machine schedulers of DISSECT-CF and comparing their

[1] `https://github.com/Ali-Alhaboby/Dissect-cf-WMS`

DOI: 10.14750/ME.2024.035

energy consumption. We used well-known workflows for our evaluation: Montage, CyberShake, LIGO and SIPHT. This makes our results comparable with future studies. In the past, they also have been used for various benchmarks and performance evaluations [36]. We have demonstrated the integration of the auto-scaling mechanisms into a larger-scale Montage workflow. Finally, we compared the experimental results of DISSECT-CF-WMS with those of WorkflowSim [19] regarding simulation accuracy and performance. We also evaluated our serverless simulation to replicate our experiments from the previous chapter.

The experimental results show that workflow researchers can investigate different PM schedulers of a given infrastructure with different numbers of VMs to achieve lower energy consumption. DISSECT-CF-WMS performs better than WorkflowSim when the number of tasks in the workflow increases. The experiments also show that DISSECT-CF-WMS is up to 295$\times$ faster than WorkflowSim and still produces equivalent results. The experimental results of the auto-scaling mechanism show that the integration has the potential to optimise makespan, energy consumption and VM utilisation compared to static deployment. The results of our serverless simulation validated the real-life experiments from the previous chapter.

Structure of the chapter: Section 3.2 summarizes the related works and briefly presents previous results. The details of the design and implementation of the DISSECT-CF-WMS simulator are given in section 3.3. We show the performance evaluation of our approach in section 3.4. In section 3.5, we conclude the chapter.

3.2 Related Works

Several simulators [19, 31, 47, 53, 68] have been developed for modelling the execution of scientific workflows on distributed platforms such as HPC, Grid, and Cloud. Some simulators [40, 54] have integrated with a particular WMS to obtain more advanced simulation.

Although several workflow simulators [14, 31, 68] exist today, they cannot meet the requirements of workflow management systems. These requirements include physical machine state scheduling, virtual machine creation details, and virtual machine placement. This behaviour does not allow analysing the impact of the various decisions made by workflow management systems at the infrastructure level. In addition, many workflow simulators [12, 19, 28, 54] do not take into account the provisioning delay of a VM in the cloud. This can have a significant impact on simulation results. This is especially true for auto-scaling, which requires VMs to be provisioned and de-provisioned while a workflow runs in the infrastructure.

Workflow integration has been demonstrated for DISSECT-CF by GroudSim and ASKALON [40]. DISSECT-CF was integrated with GroudSim to improve GroudSim's network model and cloud infrastructure simulation accuracy. This was achieved by

DOI: 10.14750/ME.2024.035

introducing internal IaaS behavioural knowledge into GroudSim using DISSECT-CF. The integration allowed ASKALON WMS to interact with the simulated cloud-like with real systems. An evaluation using a 3000-core simulated cloud showed the potential to improve ASKALON's behaviour in networking, energy metering, VM instantiation, and CPU partitioning accuracy. On the other hand, this integration caused significant additional work due to the required joint coordination between the two simulators and ASKALON. In contrast, our newly developed DISSECT-CF-WMS simulator extension relies directly on DISSECT-CF without incurring any coordination overhead. This new direct extension approach also enables the use of previously unavailable auto-scaling mechanisms that can be integrated into the execution of workflow applications on simulation infrastructures.

In [19], the authors presented the WorkflowSim simulator as an extension of the CloudSim simulator. It is designed to run scientific workflows and investigate scheduling and clustering techniques. It includes a task/job fault generator and monitor. It adds queuing/clustering delays to the workflow simulation to more accurately estimate the total execution time of the workflow. WorkflowSim does not capture all the relevant details of the system and its execution [17]. In comparison, we developed a WMS simulator on DISSECT-CF that captures the internal details of the cloud infrastructure and enables the evaluation of WMS execution on three PM schedulers. In addition, DISSECT-CF-WMS has better performance than WorkflowSim when the number of tasks in the workflow is increased. Finally, WorkflowSim does not support an auto-scaling technique or a delay in deploying VMs to the cloud, while DISSECT-CF-WMS does.

The authors introduced the WRENCH simulator in [17], which builds on SimGrid [16], a versatile, accurate, and scalable simulator. WRENCH implemented the Pegasus production WMS as a case study. Compared to WorkflowSim, it was found to be slower by a factor of $\sim$1.81 for 10,000 task workflows. This was considered acceptable since WorkflowSim's simulation results were found to be inaccurate. However, we show that our simulator approach is significantly faster than WorkflowSim. Transitively, based on the measurements of [17] measurements, we can conclude that our approach would also be faster than WRENCH.

NetworkCloudSim [28] is a CloudSim extension mainly used for simulating scheduling mechanisms. It does not support dynamic auto-scaling and provisioning delay of a VM in the cloud. In contrast, DISSECT-CF-WMS has dynamic auto-scaling that considers the start-up time of a virtual machine in the cloud.

ElasticSim [12] is a toolkit based on CloudSim for simulating workflows with support for auto-scaling techniques. It does not take into account the start-up time of a virtual machine in the cloud, which could have a major impact on simulation results. On the other hand, DISSECT-CF-WMS has auto-scaling mechanisms that consider the time needed to provision a VM in the cloud.

DOI: 10.14750/ME.2024.035

Table 3.1: *Comparison of the related workflow simulators*

Simulator	Published	Based on	Programming Language	Auto Scaling
NetworkCloudSim	2011	CloudSim	Java	-
WorkflowSim	2012	CloudSim	Java	-
GroudSim	2014	-	Java	-
ElasticSim	2017	CloudSim	Java	$\surd$
WRENCH	2018	SimGrid	C++	-

To overcome the above limitations, we developed DISSECT-CF-WMS as an extension of DISSECT-CF for analysing internal IaaS behavioural knowledge. This extension enables the evaluation of three physical machine schedulers of a given infrastructure through fine-grained energy consumption modelling. Furthermore, it can also perform large-scale workflows with good execution simulation performance. Finally, it provides an auto-scaling mechanism to dynamically provision and de-provision resources when running workflows, considering the provisioning delay of a VM in the cloud. It also provides a serverless simulation for executing scientific workflows on Lambda.

The comparison of the discussed simulators can be seen in Table 3.1. For the comparison, we listed the dependencies of the concrete simulator and the programming language, and we also indicated the year when its source code was published. Furthermore, we depicted with $\surd$ the Cloud if they have considered it in the simulators and the auto-scaling feature.

3.3 The DISSECT-CF Workflow Management System

We implemented our WMS simulation approach on DISSECT-CF, a simulator focusing on internal infrastructure. We chose DISSECT-CF because of its compact API: (i) enables easy extensibility, (ii) supports IaaS energy consumption evaluation, and (iii) enables quick evaluation of different scenarios for IaaS scheduling and internal behaviour. The APIs of DISSECT-CF support the modelling of cloud computing, network resources, job executions and file transfers. DISSECT-CF allows the definition of many types and quantities of physical machines, energy consumption properties and custom VM and physical machine schedulers. In addition, DISSECT-CF provides a virtual machine abstraction that includes migration and consolidation features. DISSECT-CF, therefore, provides all the basic abstractions required to implement classes of cloud resources relevant to the execution of scientific workflows. We developed DISSECT-CF-WMS to focus on the user side of the clouds, while DISSECT-CF

DOI: 10.14750/ME.2024.035

focuses on the internal behaviour of the IaaS systems. We extended the DISSECT-CF simulator with an extension that was built on top of the cloud system.

DISSECT-CF-WMS handles all interactions related to the execution of workflows with DISSECT-CF, e.g., transferring data, executing jobs and completing notifications. The DISSECT-CF-WMS API provides a higher-level simulation focused on WMS research. This API provides several relevant higher-level interactions with the DISSECT-CF simulator:

- To characterise the datacentre configurations for the simulated workflows, details of networks, hosts and data centre-level scheduling (e.g. VM placement policies and PM schedulers) must be provided.
- To enable parsing of workflow descriptions. This allows the loading and handling of task details and dependencies.
- To provide a custom workflow scheduling algorithm. Researchers can develop new approaches for mapping tasks to the virtual infrastructure supporting the workflow.
- To specify and set up the auto-scaling mechanism that manages the simulated virtual infrastructure hosting and running the workflow.
- To select the time to start the workflow. This helps to identify the transient behaviour of the workflow.
- To instrument the simulation for future analysis. For example, it is possible to configure the collection of details such as the total execution time of a workflow, energy consumption, resource utilisation and information about custom VM and physical machine schedulers.

The shaded part of Figure 3.1 shows the main components of the DISSECT-CF-WMS architecture. The figure also shows the main connections between the existing components of the simulator and our new WMS extensions. The figure shows how the scheduler uses virtual infrastructures to send workflow jobs. While the figure also shows that the virtual infrastructures are modelled on pre-configured clouds, our additions and their detailed connections with DISSECT-CF are explained in the following subsections.

The right side of Figure 3.1 shows the relevant components of DISSECT-CF, which manage all models for computation, storage, network and data location. The architectural novelty of our simulation extension is its use of the VirtualInfrastructure class (instead of directly interacting with lower-level components), which enables static and dynamic provisioning of VMs. Dynamic provisioning allows scaling VMs while the workflow runs. Virtual infrastructures can instantiate and terminate a specific type of VM when a user's resource needs are more dynamic and sometimes

DOI: 10.14750/ME.2024.035

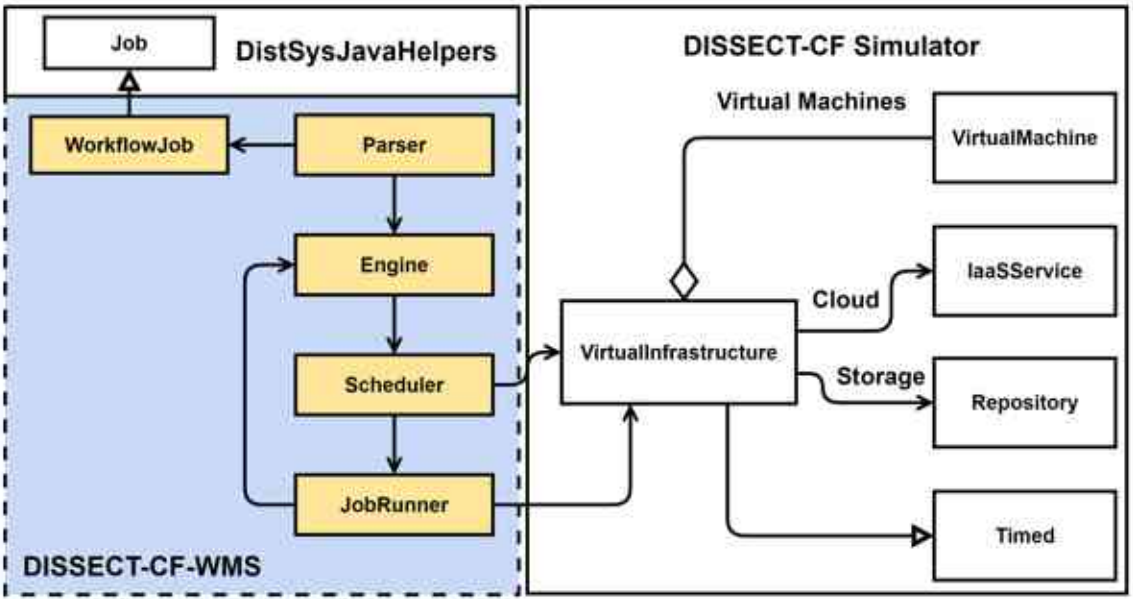


Figure 3.1: *Class diagram shows the DISSECT-CF-WMS simulator and its connection to DISSECT-CF. The DISSECT-CF simulator and Helper support the covered area in white colour.*

unpredictable. The VirtualInfrastructure class provides interfaces to all relevant internal components to create clouds, storage and virtual machines. The Timed class provides time-related notifications and enables control of the entire simulation. The IaaSService class represents a single IaaS cloud that can host the virtual machines for our virtual infrastructure and workflows. The tasks of the IaaSService are to maintain and manage the physical machines and schedule VM requests among PMs. Through the IaaSService class, DISSECT-CF supports the cloud as one of the most common execution environments, such as a commercial cloud (e.g., Amazon Web Services (AWS)[2]) and private cloud infrastructures (e.g., those managed by OpenStack[3]). The Repository class represents the storage entities in the system and is responsible for modelling data dependency. Such repositories also simulate data storage. The VirtualMachine class simulates the behaviour of a virtual machine on a physical machine.

DISSECT-CF has two types of events: time-dependent and state-dependent. First, the time-dependent events are placed in the event queue of the Timed class. The event subsystem of DISSECT-CF is used to maintain time within the simulated system. Second, the state-dependent events are fired by the entities whose states have been observed. We have used state-dependent events in our WMS that allow DISSECT-CF-WMS to be notified when a task has been completed during the execution of a

[2]`https://aws.amazon.com`
[3]`https://www.openstack.org/`

DOI: 10.14750/ME.2024.035

workflow. The DISSECT-CF-WMS simulator subscribes to all tasks to be notified when a task is completed. This allows DISSECT-CF-WMS to provide the execution time of each task within the workflow and the input/output times of the data transfer files of their data dependency.

3.3.1 Parser

The parser component reads the workflow definition from widely used DAX (Directed Acyclic Graph in XML) files (Pegasus' workflow description [21]). Parsing creates a list of WorkflowJob instances based on the workflow description files. The WorkflowJob class is an extension of the original Job class of the DistSysJavaHelpers project[4], which allows the capture of job usage metrics but lacks the dependency-related information required for workflows. Each instance of WorfklowJob stores the important information needed to process each task, such as runtime, predecessor tasks and data dependencies (input/output files).

3.3.2 Engine

After the Parser component reads the workflow definition information, the Engine component receives and processes the information by checking the predecessor tasks of each task. The Engine has information about the complete structure of the workflow. Its main task is to determine which tasks are ready for execution. A task can only be ready in two ways: (i) A task without predecessors is always ready. (ii) A task with predecessors is only ready when all its predecessors have finished their execution.

Some tasks have one or more data files that are not from their predecessor tasks. Therefore, we modelled these task inputs by transferring them from a data staging site to the selected task execution site (VM). Generally, the data staging site is a shared file system at the execution site, such as NFS, or in some cases, a file storage service for the execution site, such as Amazon S3. This site is modelled as central data storage and is used to stage data in and out for a workflow. Then, the next component will maintain task dependency constraints for managing the scheduling process.

Our engine only checks the first readiness criterion to simplify the engine's task. To ensure that we still process all ready tasks of the second criterion, we ensure that DISSECT-CF notifies the engine about the task completion for all previously detected ready tasks. When the engine receives the notification of task completion, it updates the task's successor tasks by removing itself from the predecessor list of successors.

[4] `https://github.com/kecskemeti/DistSysJavaHelpers`

DOI: 10.14750/ME.2024.035

This allows the successors to be eligible for scheduling by the workflow scheduler chosen by the simulator user.

Our WMS extension also provides different task states. Figure 3.2 shows the sequence of task states from an unavailable state (not a ready task) to a ready state when all predecessor tasks have been completed. Next, it is either in a running state if scheduled on a resource (VM) or in a waiting state if no resource is available.

One of the problems in developing a WMS is how to deal with failures. Computing resources have a small likelihood of failure during the execution of the WMS. As demonstrated via the DCF-Exercises project[5], DISSECT-CF can mimic arbitrary infrastructure failures by using a random failure generator. Our WMS extension builds on this capability by specifying each failure's cause, which allows triggering a task if its computation fails. This capability is introduced in the Engine component, which monitors the status of a task and takes appropriate action, i.e., a failed task is automatically resubmitted for execution after a timeout. For example, if a task fails, our WMS sends it to the queue and resubmits it to another computing resource. This method is part of DISSECT-CF-WMS and can be used to simulate VM failure probability and error handling capabilities for simulated workflow executions. This can be used to create more robust fault tolerance mechanisms. Finally, when the task has successfully finished its execution, it will be in a completed state.

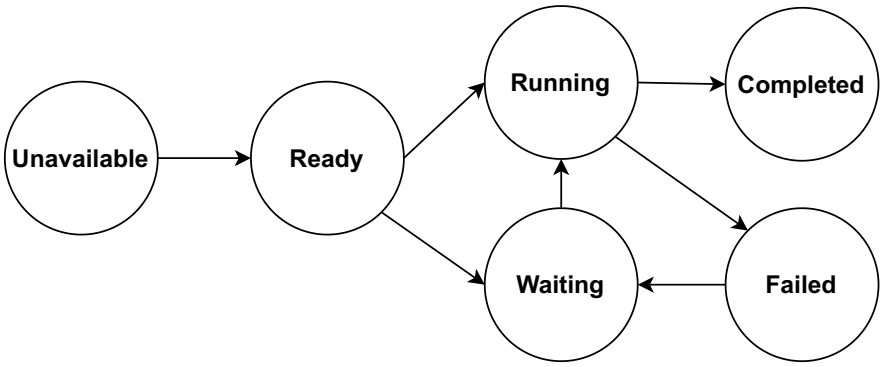


Figure 3.2: *A task state diagram.*

3.3.3 Scheduler

The scheduler receives ready tasks in a local queue from the Engine component to schedule them. In this step, we need to select appropriate resources for the ready tasks using the scheduling algorithm specified by the user. The user can implement his/her algorithm, or it can be one of the pre-implemented solutions, i.e., HEFT [67], MinMin, [9], and MaxMin, [10] (these were introduced in section 2.4). We

[5] `https://github.com/kecskemeti/dcf-exercises`

DOI: 10.14750/ME.2024.035

support two types of scheduling algorithms: static (e.g., HEFT) and dynamic (e.g., MaxMin, MinMin and DataDependency). Static algorithms start with assigning tasks to VMs in the workflow planning stage. Assignment of tasks to VMs occurs before the start of workflow execution. In this case, the Engine component is still responsible for releasing tasks whose predecessor tasks have completed execution. However, the Scheduler component assigns a task to its corresponding resource beforehand. In contrast, the dynamic algorithms start assigning tasks to VMs during workflow execution. Tasks are assigned to VMs when the tasks are ready, and VMs are free during the workflow execution phase.

We have developed the DataDependency scheduling algorithm that takes into account data transfers. It selects several ready tasks from a list of task objects stored in the data structure for execution based on the free available resources (VMs). The scheduler component is also responsible for storing information about which VM to execute each task on. This informs the JobRunner component about the location of files with data dependencies (e.g., where the predecessors stored their outputs). This step enables the actual execution of the task on a virtual machine, which is covered in the next section.

3.3.4 JobRunner

The JobRunner component is responsible for the execution of each task on a previously selected resource (VM). It transfers all files on which a task depends to the execution VM. The transfer is done using the network APIs of DISSECT-CF. If a task is assigned to the same VM executing a predecessor task, the corresponding dependency transfer does not occur. After notification of the completion of the dependency transfer, the execution of the task on the VM begins. DISSECT-CF provides the task-level resource sharing and execution model. This allows DISSECT-CF-WMS to obtain accurate and reliable information about task completion.

If necessary, JobRunner transfers the output files from an execution site to the central storage site (staging data out) after a task is completed. After completing all these activities, the JobRunner notifies the Engine component that the task is complete. This step enables the engine to schedule successor tasks.

3.3.5 Dynamic behaviour

Figure 3.3 shows the basic interaction required in our extension to execute a single workflow task. DISSECT-CF-WMS simulates the exchange of a large number of messages between its components about the state of the task.

When the process is complete, the Engine component receives the details of the workflow information in a data structure. It then forwards the ready tasks to the

DOI: 10.14750/ME.2024.035

scheduling process. Before scheduling takes place, the Scheduler retrieves information about the available resources (VMs) from the VirtualInfrastructure component of DISSECT-CF. Based on the dynamic information about resource availability, a ready task is assigned to a VM using the selected workflow scheduler. Then the JobRunner component manages the transfer of files with data dependencies to an execution site (VM) to start the execution of a task. Finally, the JobRunner component sends a task completion notification to acknowledge the engine component. This allows the successor tasks of the completed task to update their precedence conditions and be ready for scheduling. Therefore, the scheduling process continues till all tasks are scheduled. The Engine component is also responsible for completing the execution of the workflow. It counts the number of completed tasks. Once all parsed tasks have received a completion notification of execution from JobRunner, the Engine shuts down the other WMS components (i.e., the Scheduler and the Job Runner) associated with the execution of the workflow.

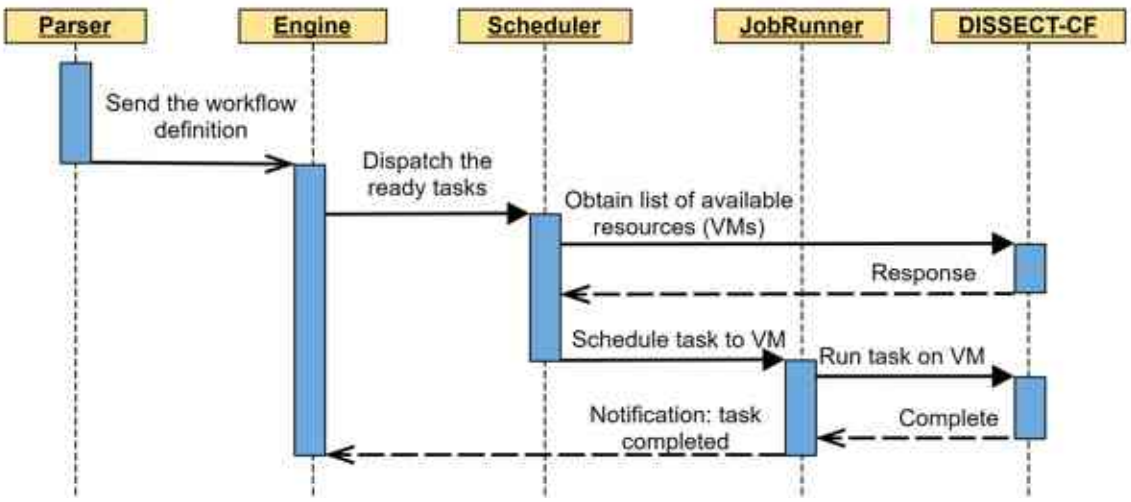


Figure 3.3: *The interactions between the DISSECT-CF-WMS components and DISSECT-CF for the task's lifecycle till it is completed.*

3.3.6 Auto-Scaling Mechanism

We integrated the DISSECT-CF-WMS simulator with the existing auto-scaling mechanisms of DISSECT-CF. We have adapted DISSECT-CF-WMS to provide auto-scaling for a workflow execution environment. We have considered the delay in provisioning a VM in the cloud, which can significantly impact simulation results. After a virtual machine is requested, it is not immediately available for use. The provisioning delay of a VM is the time it takes to be provisioned and booted on a physical host. This enables analysis of the dynamic provisioning of resources while running scientific

DOI: 10.14750/ME.2024.035

workflows in the cloud to overcome issues of under or over-utilisation of resources. The auto-scaler behind our WMS extension provides dynamic provisioning and de-provisioning of the number of VM instances based on user-selected criteria.

We have integrated our WMS into the virtual infrastructure of an auto-scaler, which is presented in section 2.7. The auto-scaler can automatically scale up or down resources based on the auto-scaling approach to better meet the demands of newly arrived tasks. We modified the JobRunner component to accommodate data transfers. Since the auto-scaled virtual infrastructure creates and destroys VMs at will, the memory of these VMs is volatile and cannot be used for long-term storage of data dependencies during workflow execution. Therefore, our approach places data files in a central data storage for staging data to and from a workflow. To avoid bottlenecking the tasks that access the same central storage to store and retrieve the data files. Therefore, we store the intermediate data on VMs during workflow execution. However, when we need to destroy a VM, we transfer the data on this VM to a central storage before destroying it. DISSECT-CF provides three basic mechanisms for auto-scaling (we discussed these in detail in section 2.7). When configuring workflow experiments, the auto-scalers can be selected, and their effects on the WMS analysed.

Auto-scaling provides a dynamic and scalable way of scheduling multiple workflows simultaneously with different virtual machine images to facilitate the execution of several tasks from various workflow applications. Users can develop novel auto-scaling policies by extending the base VirtualInfrastructure class to override its methods, such as the three mechanism classes (PoolingVI, VMCreationPriorityVI, and ThresholdBasedVI), as shown in Figure 3.4. Users can develop an approach to store their intermediate data on the VMs used for execution, but the data on a particular VM should be moved to central storage when a mechanism needs to de-provision that VM. Some users require a dynamic provisioning technique for developing some workflow scheduling algorithms that need this technique. This concept applies to algorithms that use either static or dynamic resource provisioning. This technology allows algorithms to dynamically adjust the number and type of virtual machines used to schedule jobs while workflows are running.

DISSECT-CF-WMS can query the CPU utilisation for any period during workflow execution to identify the current VM utilisation pattern. Therefore, this behaviour results in either de-provisioning some unused VMs or provisioning VMs when the current VM utilisation is high, e.g. when the three auto-scaling mechanisms use this feature (VM request, VM termination). More mechanisms could be added to reflect the environment in real life.

DOI: 10.14750/ME.2024.035

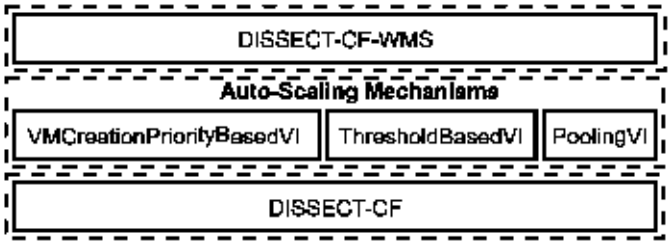


Figure 3.4: *The overview of the DISSECT-CF-WMS simulator integrated with the auto-scaling mechanisms of the DISSECT-CF simulator.*

3.3.7 A simple Model of FaaS Simulation

To develop a simple serverless workflow simulation (AWS Lambda) for executing scientific workflows, we chose DEWE v3 due to two factors: (i) it is an open-source WMS, and (ii) it already has the implementation of Lambda as its serverless execution environment. To understand our simulation model, we briefly overview DEWE v3's original and improved scheduling behaviours in the following Section 3.3.7.1.

3.3.7.1 DEWE v3

DEWE v3 runs a workflow engine on a virtual machine. When using AWS Lambda, DEWE v3 reads the workflow definition from an XML file and loads the job binaries and input files into the Amazon S3 object storage based on the information it contains. Since Lambda has a temporary storage limit of 500 MB in the execution environment, some jobs cannot be sent to Lambda due to their size. Jobs that are ready for execution (i.e., according to their precedence constraints) are scheduled into Amazon Kinesis shards.

Each shard acts as an independent queue that can send tasks to its own function instance. The Kinesis batch size determines the number of tasks a function can process in a single invocation. This can be configured before the workflow is executed. Then, the Lambda function pulls a batch of tasks from its own shard to execute them in sequence in a single function invocation. The number of running function instances and the associated kinesis shards can also be configured before the workflow is run and directly influences the maximum degree of parallelism that the execution of the workflow can have.

When a function instance starts processing a job, DEWE v3 downloads its input data from Amazon S3. When the job is finished processing, it also uploads its output data to S3 so that other jobs in the workflow can be scheduled when its input data is ready. This can result in a large amount of transfer-dependent data during workflow execution. The transfers occur between S3 and the FaaS environment and directly increase the communication costs of the workflow. Figure 3.5 illustrates the steps of

DOI: 10.14750/ME.2024.035

the original scheduling algorithm of DEWE v3.

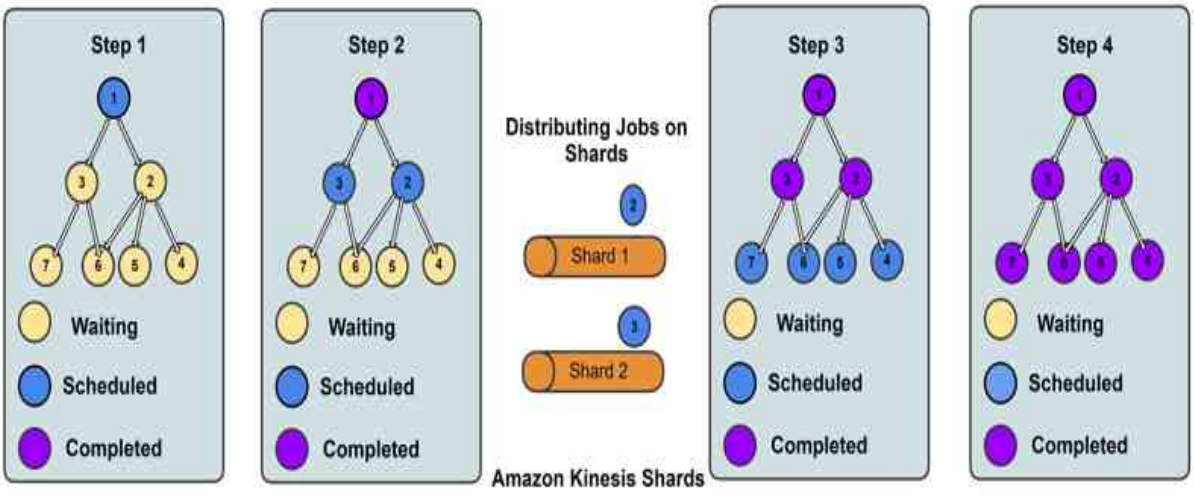


Figure 3.5: *The scheduling steps of the original algorithm with a sample workflow example.*

To avoid these transfers, we have focused on improving the scheduling algorithm of DEWE v3, which uses the Lambda platform as the execution environment. To reduce data transfers, we have considered scheduling not only the jobs that are currently ready but also their successors so that they can be executed sequentially in a single function instance. The following paragraph explains our changes in detail.

To improve the data transfers of DEWE v3, we have transferred some behaviours of the workflow management system to Amazon's Kinesis shards and Lambdas. We have taken advantage of the sequencing behaviour of shards and Lambdas. First, some jobs and their successors are scheduled for the same shard and function instance. The order of the schedule in the shard corresponds to the order of the jobs in the workflow as specified by the precedence constraints for jobs. In addition, we used the parameter *SequenceNumberForOrdering*, which guarantees the order of jobs on a shard[6]. This allows successive jobs to be executed in the same Lambda invocation without transferring output and input if these transfers are only used between those jobs. This behaviour is due to Lambda pulling a batch of jobs based on the batch size of Kinesis to execute them sequentially in one invocation. When the first job in the batch begins processing, it reads its input data from Amazon S3. Then, intermediate data (output data) is uploaded to S3, which other jobs outside batch jobs might need. Finally, Lambda also finishes processing the batch by uploading the final data files to S3. Figure 3.6 illustrates the steps of the improved scheduling algorithm of DEWE v3.

[6]`https://docs.aws.amazon.com/kinesis/latest/APIReference/API_PutRecord.html`

DOI: 10.14750/ME.2024.035

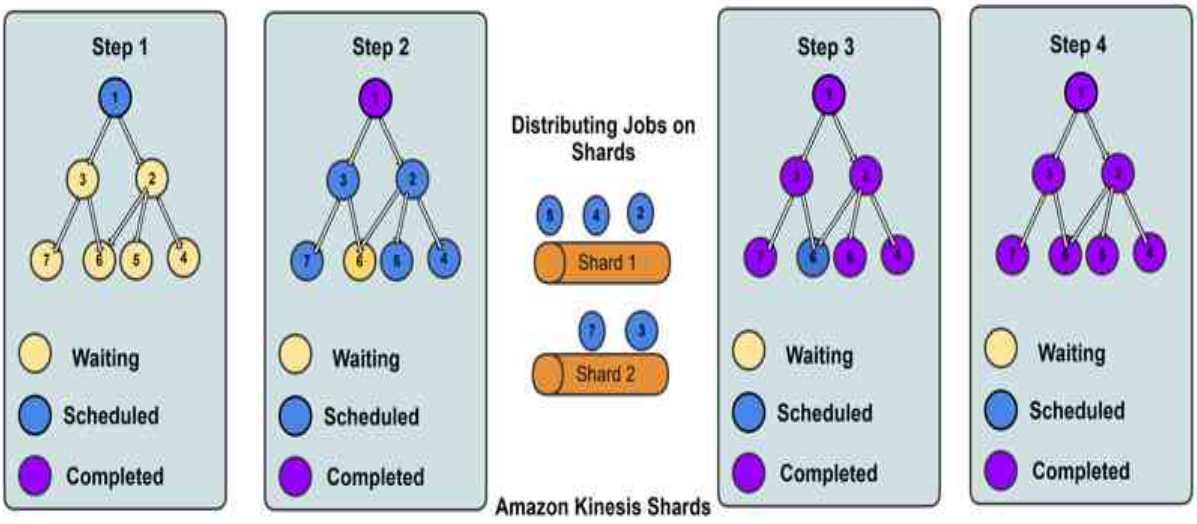


Figure 3.6: *The scheduling steps of the improved algorithm with a sample workflow example.*

3.3.7.2 The Serverless Simulation Implementation

We implemented the Function as a Service (FaaS) behaviour of Amazon Lambda to replicate our real-world experiments of serverless execution on DEWE v3. We have implemented the behaviour of the original and improved algorithms of DEWE v3, which we explained them in the previous section. First, we implemented Lambda memory sizes of 512, 1024, 1536, 2048 and 3008 MB. We assumed that 512, 1024, 1536, 2048, and 3008 memory sizes have 1, 2, 3, 4, and 5 CPU cores, respectively. We also considered each function instance as a virtual machine. Second, we limited the Lambda execution time to a maximum of 900 seconds (15 minutes). Thirdly, we implemented the batch size of the Lambda function, i.e. the number of jobs that are executed for each single function invocation. Fourthly, we implemented the number of Kinesis shards so that each shard is a specific queue for each function instance. The number of function instances depends on the number of Kinesis shards during workflow execution. Finally, we implemented Amazon S3, which is modelled as central data storage and used to stage data in and out for a workflow.

We have developed a WMS simulation to run workflows on the FaaS and IaaS simulations. We used the concept of virtual machines to run Lambda invocations on them with the Lambda constraints: Memory Limit (CPU cores), Maximum Execution Duration Limit and Temporary Storage Limit (500 MB storage space). The FaaS execution environment has only 500 MB of storage space. In addition, we used the batch size to set the maximum number of jobs that Lambda can pull from the shard to execute in a single invocation. We added a feature to calculate queuing delays that occur when scheduled jobs in Lambda invocations to shards are waiting to be

DOI: 10.14750/ME.2024.035

processed. We have added more features to calculate execution costs, total power consumption, and average utilisation of function instances.

We have developed a simple serverless workflow simulation (AWS Lambda) for executing scientific workflows with different scenarios. Our approach provides the expected time and cost of executing scientific workflows on FaaS, IaaS, and a hybrid approach combining both. The scientific community can compare the execution time and cost of workflows on IaaS, FaaS, and the hybrid approach with different configurations in the simulation. The scalability of our approach can simulate thousands of concurrent function invocations that are dynamically allocated depending on the workflow demand changes and according to the users' requirements and preferences. Our approach provides the ability to accommodate hybrid workloads using FaaS and IaaS in a single simulation. If the function invocation fails to execute a batch of jobs, these jobs are sent to IaaS to be executed on the available resource (VM).

3.4 Evaluation

We demonstrate the capabilities of DISSECT-CF-WMS using the following evaluation experiments. First, we evaluated how the pre-existing three physical machine schedulers influence the energy consumption of various workflows. Second, we compared the simulation of DISSECT-CF-WMS with WorkflowSim regarding simulation accuracy and performance. Third, we have shown the advantages of using the autoscaling mechanisms of DISSECT-CF-WMS to optimise makespan, energy consumption and VM utilisation over static provisioning. Finally, we also evaluated our serverless simulation to replicate our real-world experiments of serverless execution on DEWE v3. The simulations were run on a laptop with 12 CPUs of Intel Core i7-8750H CPU @ 2.20GHz, 16GB RAM and 119GB SSD.

All experiments were evaluated with synthetic workflows derived from the Montage (astronomy), CyberShake (earthquake science), LIGO (gravitational physics) and SIPHT (biology) applications, taking into account data transfers. These workflows are available[7].

To simplify the configuration of the simulated cloud, we used the DCCreation class from DISSECT-CF. We configured the simulated infrastructure for our WMS experiments by setting up a homogeneous cloud with 100 physical machines (each configured with 32 CPU cores, 256 GiB of memory and 256GB of storage, and a linear power model ranging from an idle power draw of 296 watts to a maximum power draw of 493 watts) and also configured central data storage of 36 TB. The machines and central storage were simulated to be connected via a single switch (we set the bandwidth between the machines and the switch to 2 Gbit). All created

[7] `https://github.com/wrench-project/pegasus/tree/master/examples/evaluation/scalability`

DOI: 10.14750/ME.2024.035

physical machines are connected via a cloud-level network. Three physical machine schedulers potentially control the physical machines.

To evaluate the energy efficiency, we used the IaaSEnergyMeter class from DISSECT-CF, which allowed us to monitor the energy of the entire IaaS system generated by the DCCreation class. We set up our energy metre to monitor the entire cloud and collect energy-related details in every simulated hour. In addition, we used the HourlyVMMonitor class, which can monitor the utilisation of each VM's CPU at an hourly rate.

We collected the data centre metrics after the first task of the workflow was started. We instrumented the following simulation experiments to capture the following metrics: (i) the makespan (total workflow execution time), i.e., the start time of the first task to the completion time of the last task, (ii) the average VM utilisation, i.e., the average of the hourly reports for each VM during the complete execution of a workflow, and (iii) the total energy consumption of the data centre in kilowatt hours (kWh) as reported by the IaaSEnergyMeter.

3.4.1 Utilisation of Internal Cloud Infrastructure Details

We configured a virtual infrastructure with a static number of VMs (in a single experiment, we set the number of VMs between 30 and 100; all VMs were homogeneous regarding the number of CPU cores and memory). We used FirstFitScheduler as a VM scheduler and the DataDependency algorithm as a task scheduler on VMs. The FirstFitScheduler is a VM scheduler that implements one of the simplest VM schedulers. It places each VM at the first PM that would actually accept it. We ran each static virtual infrastructure on the cloud mentioned above, but we replaced the schedulers for the physical machines with the three offered by the simulator: (i) AlwaysOnMachines (AOM), (ii) SchedulingDependentMachines (SDM) and (iii) MultiPMController (MPMC). First, AlwaysOnMachines ensures that all PMs are controlled to always remain on. Second, SchedulingDependentMachines increases or decreases the power of the PM set according to the requirements of the VM scheduler (this scheduler changes the power of the PM set by one PM at a time). Finally, MultiPMController is very similar to SDM but immediately increases the number of machines needed to run the current infrastructure (i.e. if four newly powered-on PMs are needed to host the current demand of VMs, all four are powered on immediately). We set a linear power model for physical machines created by DISSECT-CF-WMS, which assumes that power consumption depends on the degree of use of the CPU, ranging from an idle power consumption of 296 watts to a maximum power consumption of 493 watts. We recorded the power consumption for DISSECT-CF-WMS from the start time of the first task to the completion time of the last task of the workflow.

Figures 3.7, 3.8, 3.9 and 3.10 show the collected energy consumption for each

DOI: 10.14750/ME.2024.035

experiment of DISSECT-CF-WMS when running 1000 tasks each of the Montage, CyberShake, Sipht and LIGO workflows. With a small number of 30 VMs (4 cores) using only slightly less than 4% of the total infrastructure, the MPMC and SDM schedulers have much better energy consumption than the AOM scheduler (i.e., they consume more than 11 $times$, 15 $times$, 7 $times$, and 8 $times$ of energy for the same computation of the Montage, CyberShake, Sipht and LIGO workflows, respectively). This pattern repeats (with smaller advantages) for almost all larger VM numbers, except when the VMs use the entire infrastructure. In all cases, AOM's strategy of switching on all machines regardless of workload pays off, as it makes all VMs available for workflow at the earliest opportunity. In contrast, the SDM and MPMC schedulers significantly reduce energy consumption. SDM uses a strategy of switching on all machines in the data centre one by one (at a time), resulting in a long overall simulation time due to the provisioning delay, as shown in Figure 3.11. The MPMC policy, on the other hand, immediately switches on the number of machines needed for the current operation of the infrastructure. Static VM allocation policies for workflows are unsuitable for data centres using a PM scheduler such as SDM.

First, AOM has the same energy consumption patterns for all workflow applications because it never considers switching off machines, and thus it results in energy consumption for the entire infrastructure, even for the PMs that do not host VMs. In the case of AOM, increasing the number of VMs also decreases the makespan, which reduces energy consumption. Second, all PM schedulers have the same energy consumption when using the entire infrastructure, as they use all machines. Moreover, the MPMC and SDM schedulers have similar patterns for the Montage and CyberShake applications. However, the Sipht and LIGO applications have reduced energy consumption by increasing the number of VMs because they have not used all the statically created VMs at all times. In the experiments with 30, 40 and 50 VMs, the workflow scheduler reused VMs during workflow execution. In the case of the Sipht experiment with 100 VMs, however, which only uses a maximum utilisation of the VMs of 70%, because there are 64 jobs in the second phase of Sipht, while the number of VMs is 100 and therefore 36 VMs are unused. This leads to a significant underutilisation of resources in this phase. The idle time in these resources leads to the highest energy consumption. This behaviour is also repeated in the last two phases of the Sipht workflow. In addition, some tasks in Sipht have significant differences in their runtimes, so the time difference can be up to 19$\times$. In the LIGO experiments, this pattern is repeated, but the time difference can be up to $3\times$, resulting in lower energy consumption.

If we compare the experiments from the cloud users' point of view, the results show the advantage of the AOM and MPMC schedulers. As our base WMS waits for all statically allocated VMs to start up, the VMs behind our workflows can start faster thanks to AOM's always-ready physical machines. This reduces VM provisioning time,

DOI: 10.14750/ME.2024.035

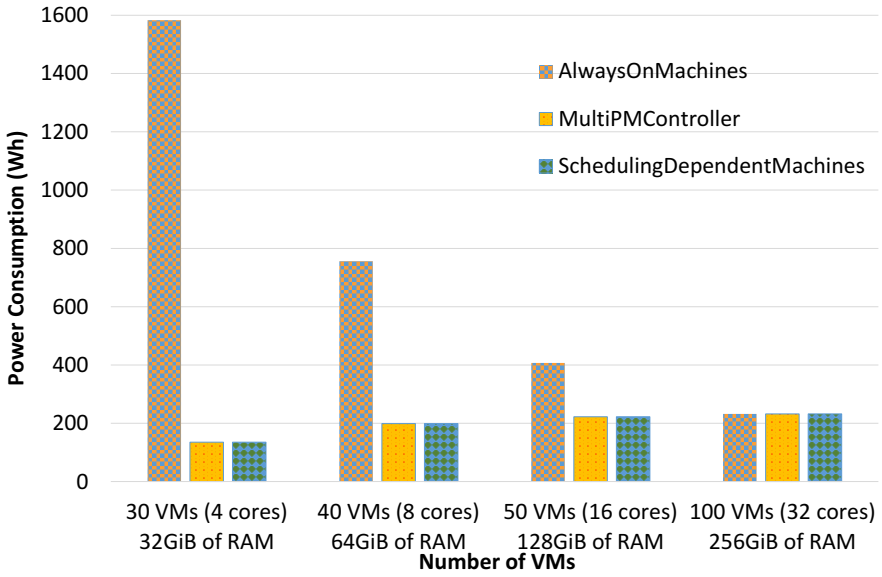


Figure 3.7: *The total power consumption of PM schedulers for the Montage workflow on DISSECT-CF-WMS with different numbers of VMs.*

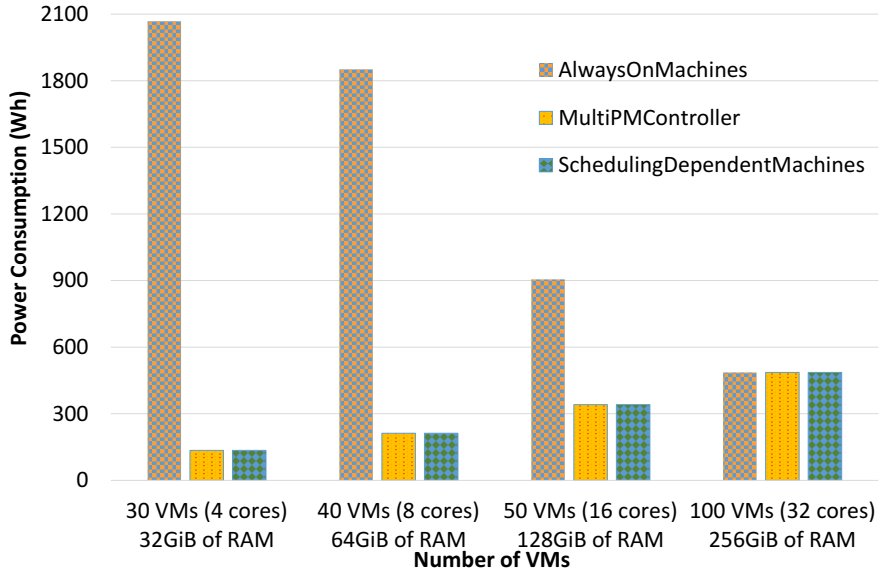


Figure 3.8: *The total power consumption of PM schedulers for the CyberShake workflow on DISSECT-CF-WMS with different numbers of VMs.*

as shown in Figure 3.11. Note that despite AOM's significant energy penalty, the improvements in provisioning time are equally significant. The weakness of the SDM strategy is also evident in the waiting time. Our WMS has to wait significantly longer for the requested VMs to be ready before assigning tasks to them. The waiting time difference can be as high as 17× as shown in Figure 3.11. The differences are mainly

DOI: 10.14750/ME.2024.035

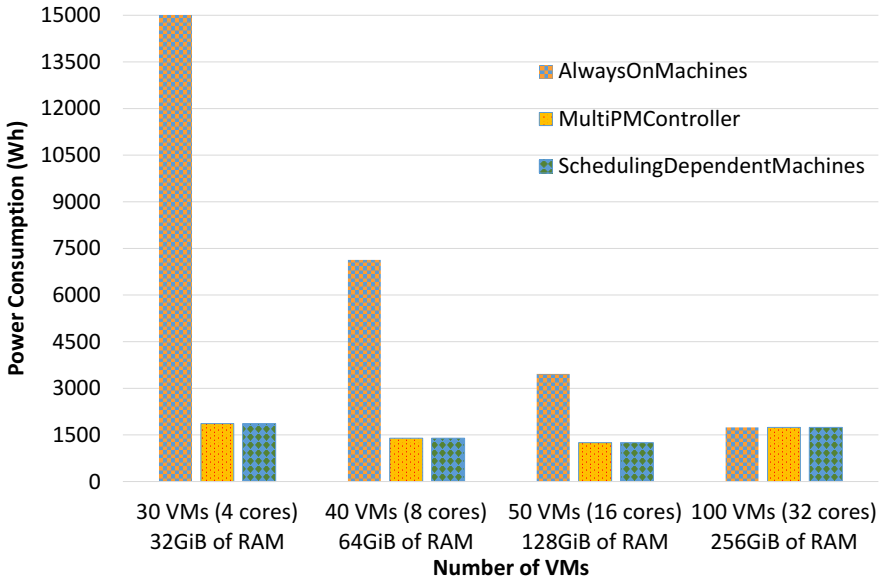


Figure 3.9: *The total power consumption of PM schedulers for the Sipht workflow on DISSECT-CF-WMS with different numbers of VMs.*

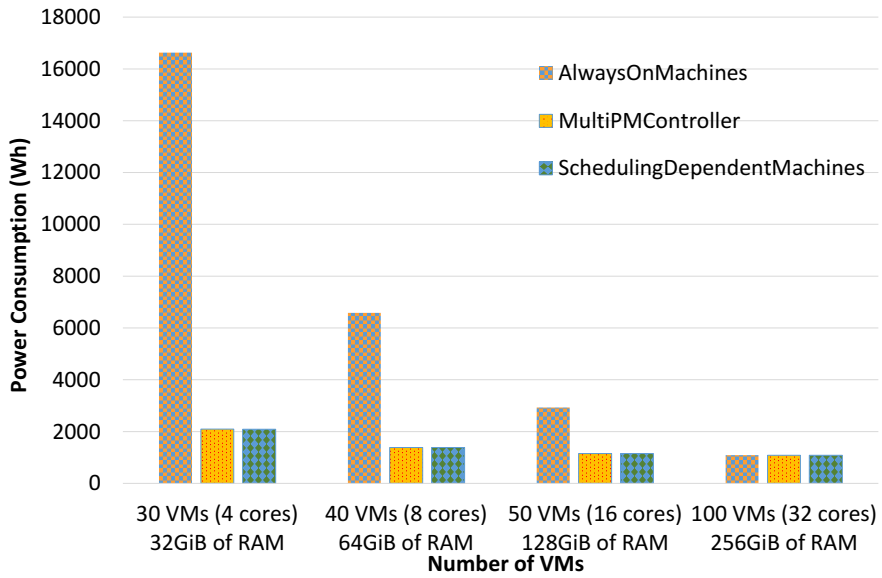


Figure 3.10: *The total power consumption of PM schedulers for the LIGO workflow on DISSECT-CF-WMS with different numbers of VMs.*

because SDM is very slow in starting machines (one VM at a time). As a result, the execution of the entire workflow is delayed with fewer physical machines turned on (but those few are turned on for a significantly longer time, as shown by the provisioning times in Figure 3.11). These differences show that switching on all PMs required for the workflow is advisable for dedicated private cloud infrastructures.

DOI: 10.14750/ME.2024.035

This way, we get the results back the fastest and also do not consume too much energy during the runtime of the workflow, like the MPMC scheduler. Moreover, all workflows in SDM started executing the tasks after all required PMs were switched on. Therefore, SDM and MPMC consume the same energy despite having different strategies. Thus, DISSECT-CF-WMS can offer insight for analysing different workflow execution scenarios and instrumenting the execution environment to gain insight into the impact of the chosen infrastructure configuration.

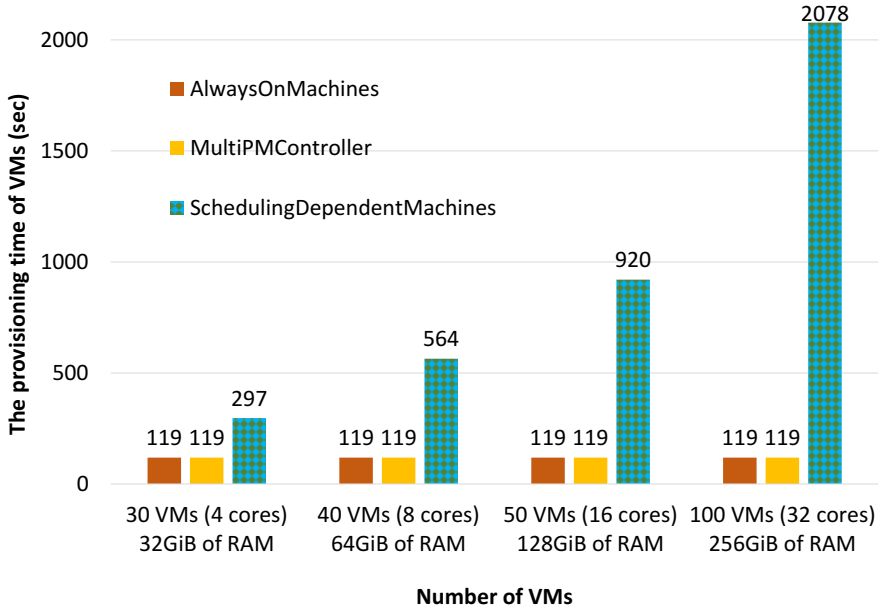


Figure 3.11: *The provisioning time of VMs for three PM schedulers on DISSECT-CF-WMS with different numbers of VMs.*

3.4.2 Simulation Times

We have demonstrated the benefits of a WMS simulation extension for evaluating WMS behaviour. We move on to the evaluation of the core functions of the WMS. We have compared the performance and accuracy of the simulation results of our DISSECT-CF-WMS with version 1.1.0 of WorkflowSim, using the same laptop as mentioned above. We chose WorkflowSim because it is an open-source workflow simulator providing a higher workflow management layer. DISSECT-CF-WMS does not use logging mechanisms, but we printed the execution details as messages. To ensure a fair comparison, WorkflowSim's logging mechanisms were disabled. We ran two experiments, one in each simulator, with exactly the same settings. First, we ensured the simulated data centres had the same characteristics. Again, we used the same cloud mentioned earlier. We requested a static VM configuration that occupied the

DOI: 10.14750/ME.2024.035

entire data centre: 100 virtual machines with 32 cores each. For our WMS, we used FirstFitScheduler as the VM scheduler, AOM as the PM scheduler and DataDependency as the scheduling algorithm. For WorkflowSim, we used DATA as the scheduling algorithm, LOCAL as the local file system for storing the data dependency files and the time-shared model as the policy for VMs and jobs. We have evaluated both simulators with synthetically generated Montage workflows of different sizes (the number of tasks ranged from 1K to 15K).

We compared the total execution times reported by both simulators for all the workflows. We have obtained very similar execution times in both simulators. The difference between the two had a mean absolute percentage error (MAPE) of less than 0.16%. This difference in execution time is because our workflow scheduler in DISSECT-CF-WMS assigns tasks to VMs slightly differently than the approach taken by WorkflowSim. As a result, the dependent data's transfer time may differ. Despite the more accurate and detailed simulation (i.e., we provide more insight into the internals of the data centre behind the workflow), DISSECT-CF-WMS delivers the results in significantly less time. Figure 3.12 illustrates the performance differences between the simulators.

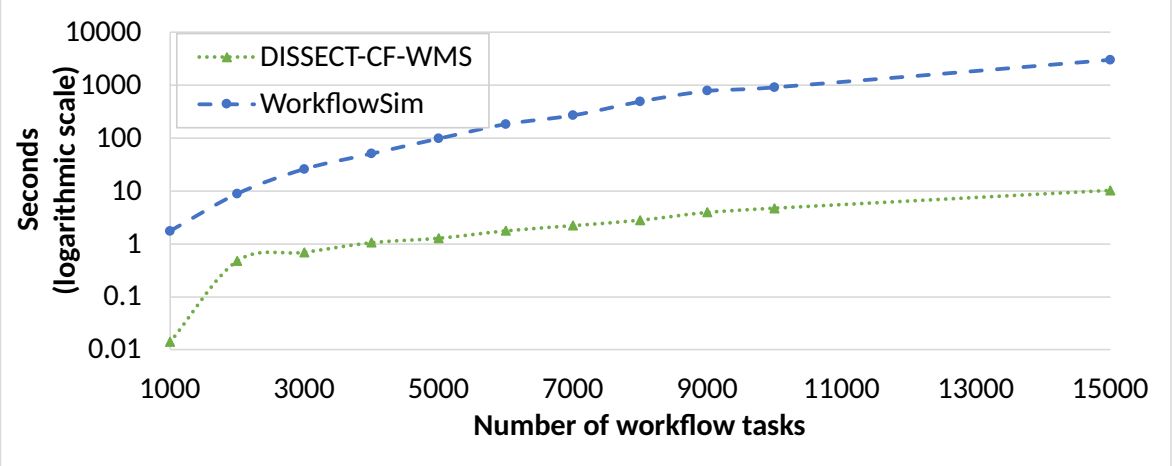


Figure 3.12: *The simulation time of the DISSECT-CF-WMS and WorkflowSim simulators with different numbers of Montage workflow tasks.*

We see that our measurements of the real duration of the simulation show that the performance advantage of DISSECT-CF-WMS is between 18 and 295× (i.e., we can get to the same quality results at most two orders of magnitude faster). Moreover, WRENCH took 13 minutes to simulate a Montage workflow with 10,000 tasks[15], while DISSECT-CF-WMS took about 5 seconds to simulate the execution of the same workflow. WorkflowSim builds on CloudSim, which uses a process-based paradigm where each entity in the system has its own thread, resulting in poor scalability as the number of entities in the system grows [56]. DISSECT-CF, however, requires

DOI: 10.14750/ME.2024.035

only one simulation thread (instead of one thread per entity). As a result, our WMS outperforms WorkflowSim, as shown in Figure 3.12.

3.4.3 Simulation versus Execution

In this experiment, we compared our simulation result to an existing execution of a real-world Pegasus workflow (Montage-2.0) on the AWS-m5.xlarge platform to validate the simulation environment [15]. The Montage workflow contains 1240 tasks, and we compared the real execution of five traces to the simulated one. We replicated the identical execution environment, which performs similarly to AWS-m5.xlarge instances. The execution environment includes a submission node that runs Pegasus and DAGMan and four worker nodes (4 cores per node with a shared file system). In these instances, the bandwidth between the data node and the submit node was 0.44 Gbps, while the bandwidth between the submit and worker nodes was 0.74 Gbps and 1.24 Gbps, respectively. Figure 3.13 depicts Gantt charts of the real execution, whereas Figure 3.14 depicts the simulated execution. On the vertical axis, task executions are shown as a line segment on the horizontal time axis, covering the time interval between the task's start and end times. Different kinds of tasks are indicated in different colours. We have tried to assign the same colours to these task types as in the real execution. The average time for real-world execution was 2911.8 seconds, whereas the average time for simulated execution was 2980 seconds. The results of the experiment demonstrate that the scheduling and execution of the simulated workflow are similar to the actual workflow execution. The runtime difference of 68.2 seconds is due to the errors of the prediction service utilised for choosing the activity and file transfer execution timings in the simulation, which inaccuracy is within a 3% range.

All tasks of the same type in this workflow have the same priority and are independent. For example, the shapes of the yellow areas differ in the two figures. The implementation-based behaviour of the workflow scheduler explains these differences. During the execution of the workflow, it is often possible to select several ready tasks for execution, e.g., groups of independent tasks on the same workflow level. If the number of computing resources, n, is less than the number of ready tasks, the scheduler immediately executes n-ready tasks. In most WMSs, these tasks are selected from the first n tasks returned during iteration through the data structures in which the task objects are placed. To create an identical replica of a WMS, it needs to develop and use the same data structures as the real implementation. Depending on the data structures, languages, and/or libraries used, this can be tedious or impossible. In this Pegasus case study, the real DAGMan scheduler uses a custom priority list to hold ready tasks, while our simulation version stores workflow tasks in a Java hashmap indexed by task string IDs. The consequence is that the real sched-

DOI: 10.14750/ME.2024.035

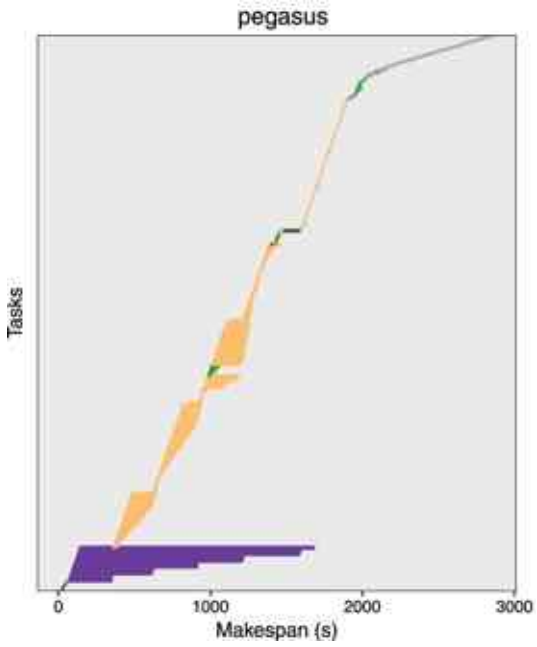


Figure 3.13: *Task execution Gantt chart for sample real-world ("pegasus") execution of the Montage-2.0 workflow on the AWS-m5.xlarge platform [15].*

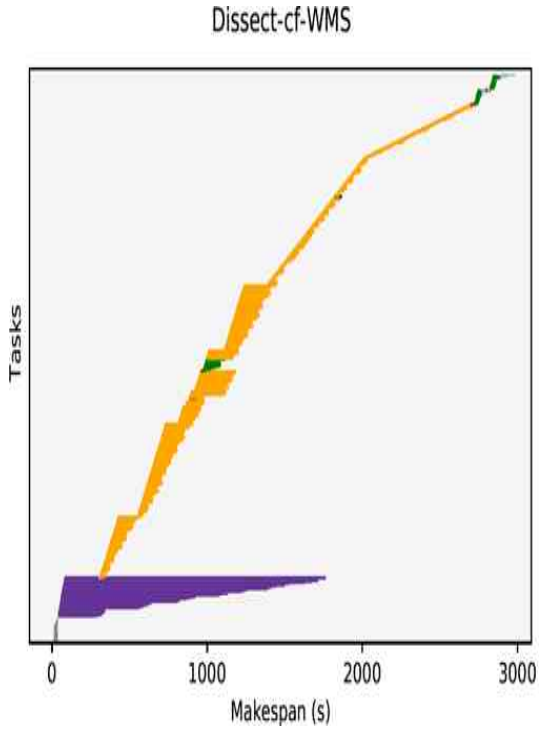


Figure 3.14: *Task execution Gantt chart for simulated Dissect-cf-WMS executions of the Montage-2.0 workflow on the AWS-m5.xlarge platform.*

uler, when selecting the first n-ready tasks, generally selects different tasks than the simulated version of the scheduler. The differences that can be seen in Figure 3.13 and 3.14 can be attributed to this factor.

3.4.4 Auto-Scaling Mechanism

We focused on demonstrating the benefits of the auto-scaling mechanisms behind DISSECT-CF-WMS. We re-ran our large-scale (15K tasks) Montage workflow. We used the same cloud we mentioned earlier. We compared the dynamic VM allocation strategies of the different auto-scalers with the completely static virtual infrastructure allocation (thus allowing a comparison to the previously acquired statically allocated Workflowsim Scenario). In the static scenario (dedicated cluster), we set up 50 virtual machines with two cores each before the workflow executes and kept all VMs until the end. For this experiment, we used FirstFitScheduler as the VM scheduler, MPMC as the PM scheduler, and the DataDependency algorithm as the task scheduler. DISSECT-CF also simulates a single repository for a specific type of virtual appliance from which all VMs can be derived. In this experiment, we modified the Pooling VI to

DOI: 10.14750/ME.2024.035

ensure the efficiency of the auto-scaling mechanism. First, we adjusted the pooling VI to have 50 VMs with two cores each at the beginning of the workflow execution since Montage has 12,495 tasks in the first and second phases. The structure of Montage is shown in Figure 2.3. Second, we set the threshold for pooling VI to 80 VMs to reduce the cost while maintaining the makespan. Finally, the number of VMs is reduced to two if the single-threaded tasks of the Montage workflow are executed sequentially. In this case, one VM is used while the second is idle because Pooling VI is designed to have a certain number of completely unused VMs available for executable jobs.

Figure 3.15 shows the results of executing the workflow. Pooling VI has the shortest total execution time compared to a dedicated cluster and the other auto-scaled virtual infrastructures. In terms of VM resource utilisation, Figure 3.16 also shows that pooling VI has the best average VM utilisation across all auto-scaling mechanisms and static 50 VMs. This is because pooling VI has been configured to always keep one VM ready in the virtual infrastructure (so this is a compromise between the fully static and dynamic scenarios that the others implement). It is also worth noting that Pooling VI follows an almost static allocation of VMs, while the other two approaches frequently destroy and recreate VMs (in fact, they only reuse VMs for about six tasks before discarding them). These approaches thus significantly increase execution time, as most workflow tasks initially have no VMs to execute and must wait for their respective VMs to come to life. It should also be noted that the additional transfers required for staging data also lengthen execution, unlike the static VM allocation approach. In addition, the VMs access the same central storage to read and write the data dependency files. Montage is a data-intensive scientific workflow. To avoid bottlenecking the tasks that access the same central storage to store and retrieve the data files. Therefore, we store the intermediate data on VMs during workflow execution. However, when we need to destroy a VM, we transfer the data on this VM to a central storage before destroying it.

A concept similar to Amazon EC2 is being considered, where VMs are rented on demand and charged hourly, with partial hours rounded up to the next full hour. Pooling VI reduced total billed hours by 41.5% compared to the dedicated cluster with 50 VMs, as shown in Figure 3.17. The Montage workflow consists of six single-threaded tasks executed sequentially, as shown in Figure 2.3, with a total execution time of about 4.5 hours. As a result, when VMs were statically allocated, only one VM was used for 4.5 hours, while the other VMs were idle due to the single-thread tasks. Another consideration was related to Pooling VI, which describes the ability of mechanisms to allocate many VMs efficiently (see Figure 3.18). Static provisioning is inefficient when the number of VMs remains constant over time. In this case, the scheduling algorithm does not provide a way to increase or decrease the number of VMs in response to a dynamic workload of workflows. In Figure 3.19, Pooling VI reduced energy consumption by more than 82% compared to static allocation. In

DOI: 10.14750/ME.2024.035

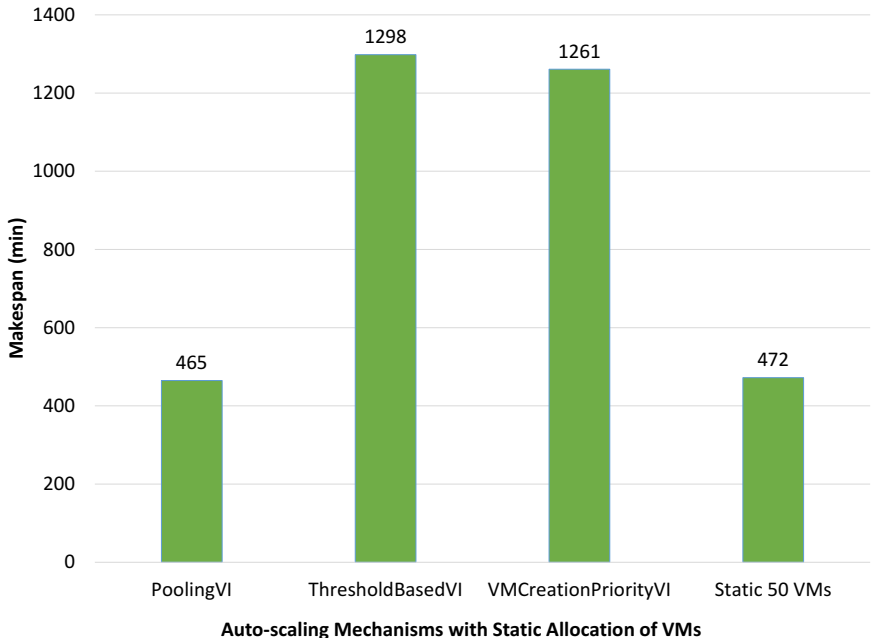


Figure 3.15: *Makespan of auto-scaling mechanisms and static 50 VMs.*

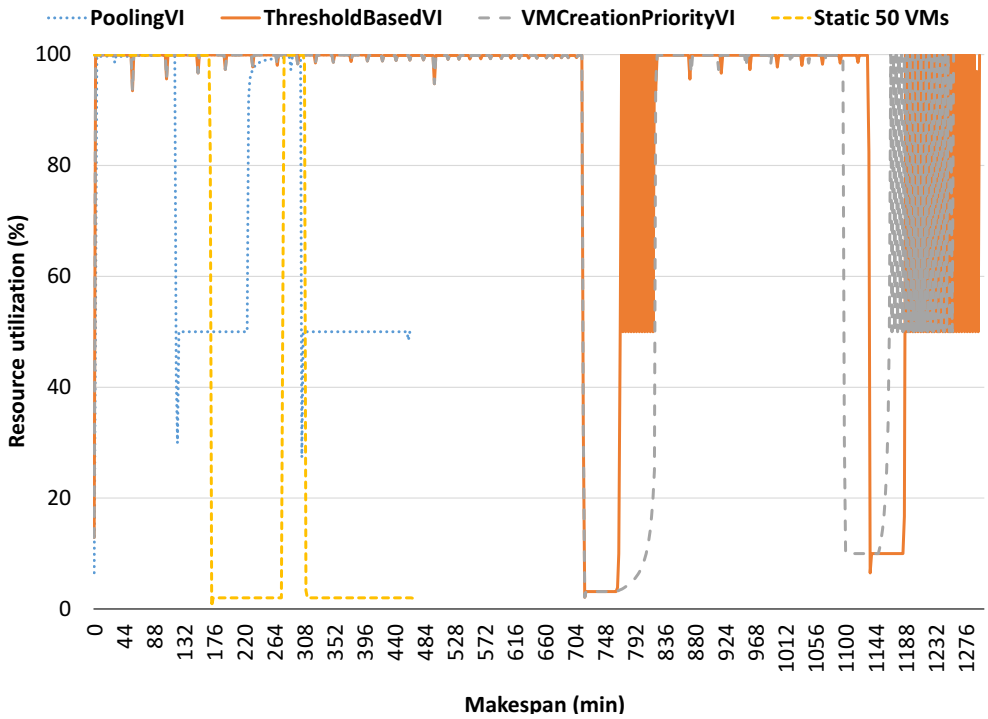


Figure 3.16: *Resource consumption patterns of auto-scaling mechanisms and static 50.*

addition, Pooling VI reduced energy consumption by about 54% compared to the other auto-scaling mechanisms. Although Pooling VI has used the most significant

DOI: 10.14750/ME.2024.035

total number of VMs compared to static allocation and the other mechanisms (see Figure 3.20), it has the lowest total number of hours billed, as shown in Figure 3.17.

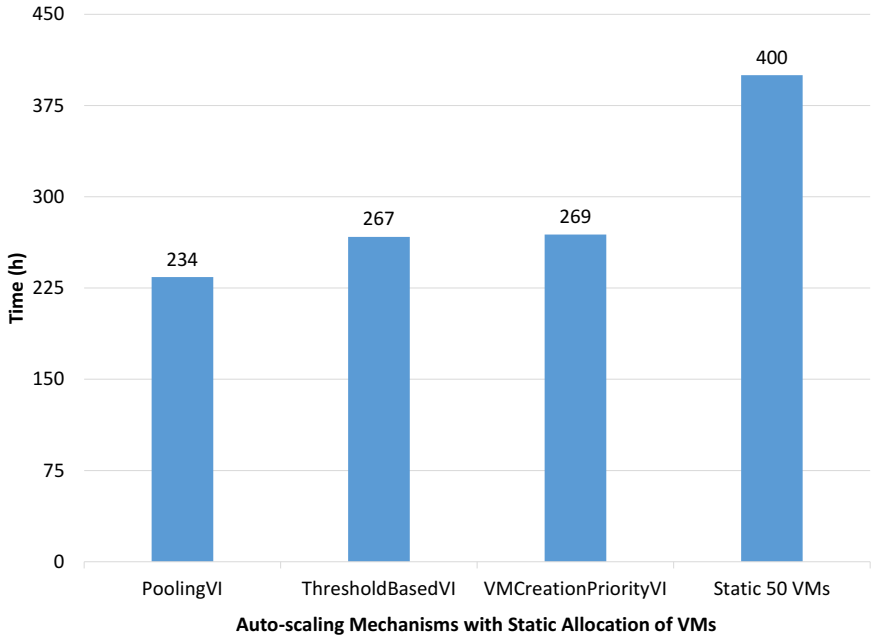


Figure 3.17: *The total accounted for hours of virtual machines of auto-scaling mechanisms and static 50 VMs.*

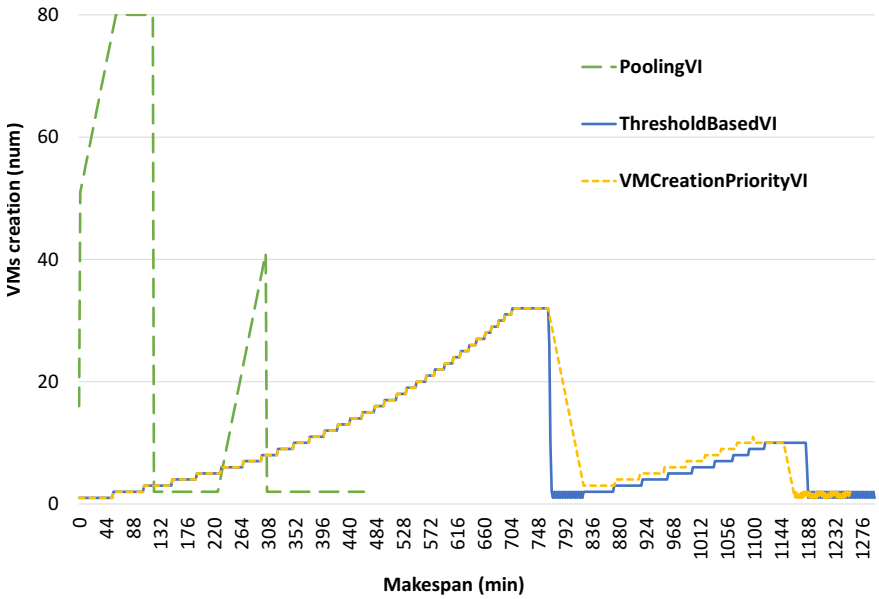


Figure 3.18: *The virtual machines creation patterns of auto-scaling mechanisms.*

DOI: 10.14750/ME.2024.035

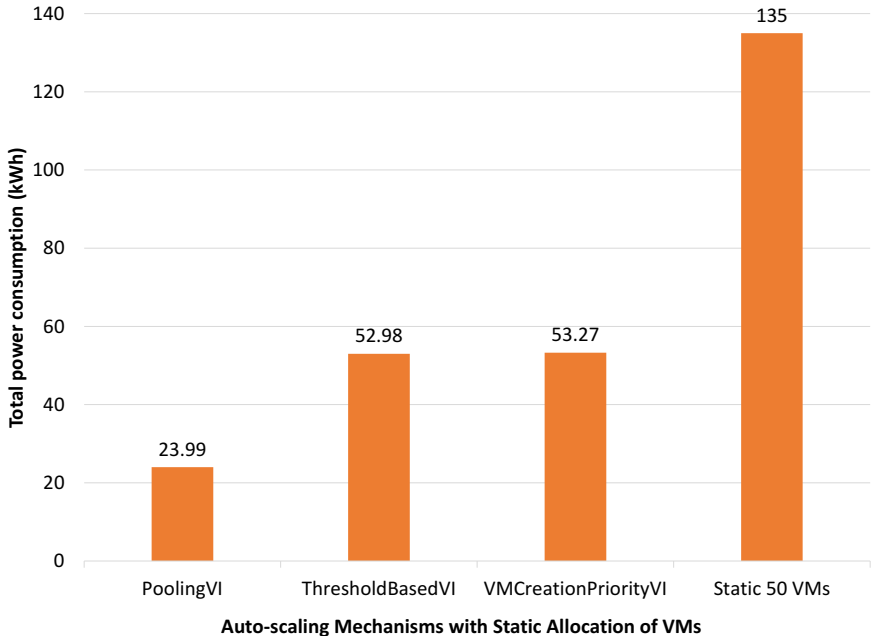


Figure 3.19: *The total power consumption (kWh) of auto-scaling mechanisms and static 50 VMs.*

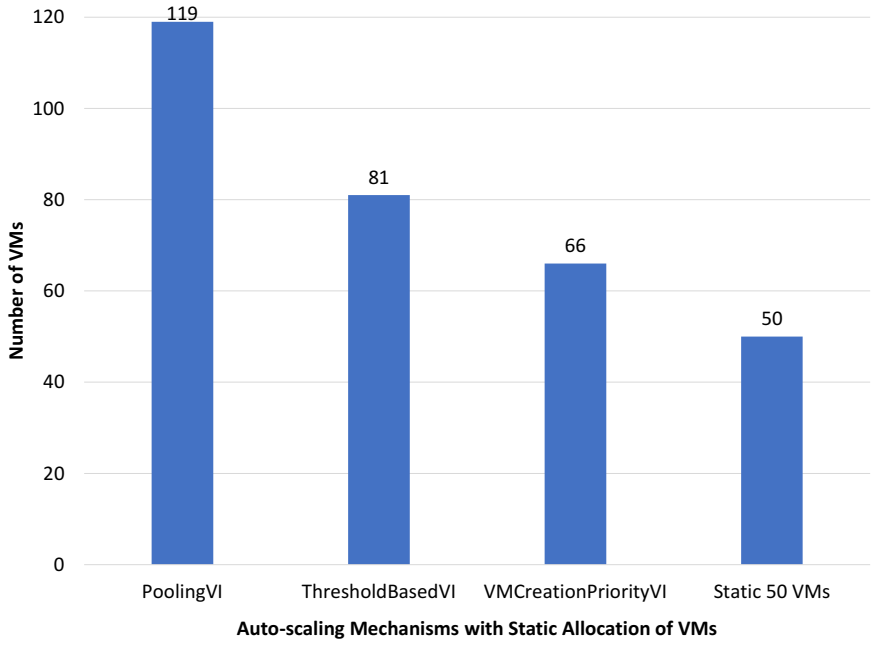


Figure 3.20: *The total accounted for the number of virtual machines (2 cores each) of auto-scaling mechanisms and static 50 VMs.*

DOI: 10.14750/ME.2024.035

3.4.5 The FaaS Workflow Experiments

The FaaS simulation model aims to reproduce what we observed in our real-world experiments of serverless execution on DEWE v3. Also, we obtain the same results from the simulation as from the real-world experiments. Real-world experiments are good to start with, but they cost so much money, and we cannot scale them up. However, we have implemented the original and improved algorithms of DEWE v3, which we explained in Sections 3.3.7.1 and 3.3.7.2.

3.4.5.1 Real-World Experiments

We evaluated the improved and original algorithms of DEWE v3 with a 6.0-degree Montage workflow considering data transfers between jobs. The 6.0-degree Montage workflow has 8,586 jobs with a data dependency size of 38GB. All the jobs are executed on Lambda except the mAdd jobs executed on a single virtual machine. The VM is needed because the size of the input/output files of mAdd exceeds the temporary storage space offered in a single Lambda function invocation. The configurations of the experiment are as follows:

1. Lambda Memory size was 3008 MB
2. Lambda execution duration was 900 seconds.
3. The batch size of the Lambda function was 20.
4. The number of Kinesis shards was set to 30.
5. The virtual machine was t2.xlarge, which has the following properties: 16 GiB of memory and four vCPUs.

The makespan of the improved algorithm is 1001 seconds, while the original algorithm was 1109 seconds. The improved algorithm reduced the total execution time of the large-scale workflow by about 10% compared to the DEWE v3 original algorithm. Thus, this experiment shows that our improved algorithm benefits larger-scale workflows.

3.4.5.2 Simulation Experiments

Next, we evaluated both the original and the improved algorithms with scientific workflows considering data transfers between jobs. The configurations of the experiment are as follows:

1. The Lambda Memory sizes were 512, 1024, 1536, 2048 and 3008 MB
2. The Lambda execution duration limit was limited to 900 seconds.

DOI: 10.14750/ME.2024.035

3. The batch size of the Lambda function was 20.
4. The number of Kinesis shards was set to 30.
5. One VM consisted of 4 CPU cores and 8 GiB of memory.

We used the same workflow applications that were run in the previous experiments. We set the speed for each CPU core to 1000 MIPS in the simulation. The bandwidth between FaaS and Amazon S3 was set to 1 Gbit. The scientific workflows have about 1000 tasks from the Montage, CyberShake, Sipht and LIGO workflows. Figures 3.21, 3.22, 3.23 and 3.24 show the total execution time (makespan) of Montage, CyberShake, Inspiral (LIGO) and Sipht workflows, respectively.

In the case of Montage, the improved algorithm schedules jobs with priority constraints to be executed in a single function invocation. Therefore, successor jobs can use the output files generated by their predecessor job in the same invocation. The improved algorithm has the potential to reduce workflow execution time by more than 10% compared to the original DEWE v3 algorithm, as shown in Figure 3.21. This demonstrates the validity of our work in the previous section. We excluded the workflow jobs (namely mAdd) from running on Faas because of the expected large dependency files and their runtimes. Consequently, all mAdd jobs were run on the VM.

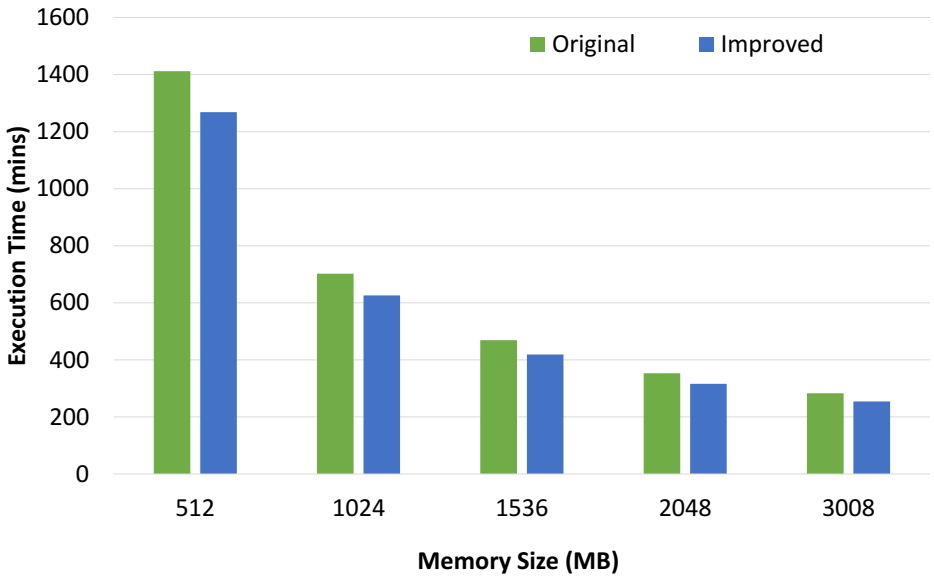


Figure 3.21: *The makespan of the two algorithms with Montage workflow running on different Lambda memory sizes.*

In the case of CyberShake, the improved algorithm reduced the makespan by 15% compared to the original DEWE v3 algorithm, as shown in Figure 3.22. Our improved

DOI: 10.14750/ME.2024.035

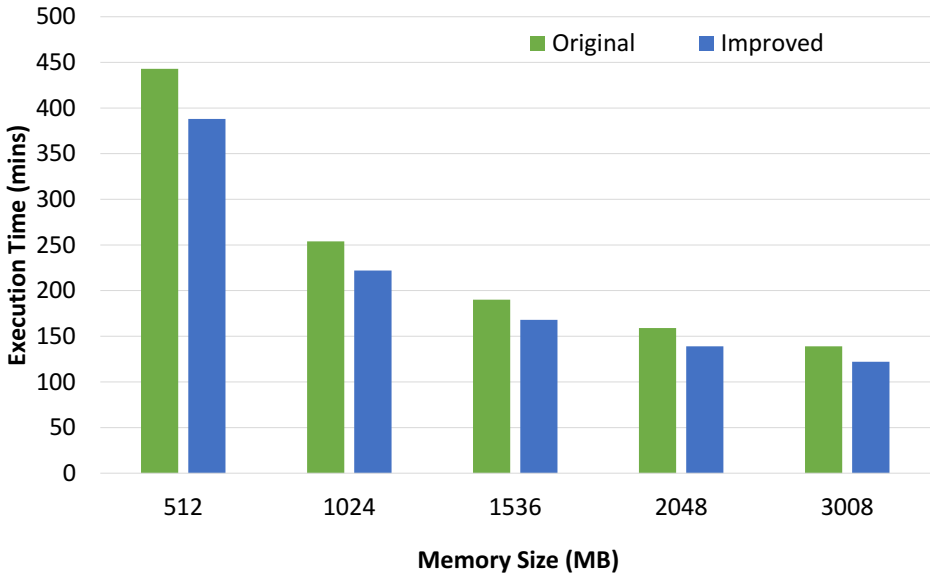


Figure 3.22: *The makespan of the two algorithms with CyberShake workflow running on different Lambda memory sizes.*

algorithm relies on two important factors for workflow execution. First, its approach depends on the workflow structure where jobs with precedence constraints can be scheduled with their predecessor job in a single function invocation. For example, in CyberShake, there are many jobs that have a single predecessor job that they can schedule with their successor. Secondly, the reduction in makespan depends on the data size of the data dependencies. For example, the average data size of CyberShake is 102 MB.

In the case of Inspiral (LIGO), the improved algorithm reduced the makespan by 11% compared to the original DEWE v3 algorithm, as shown in Figure 3.23. This pattern repeated (with almost gradual decreases) the same as for CyberShake. In LIGO, there are many jobs that have a single predecessor job that they can schedule with their successor at the same time. Furthermore, LIGO's average data size is 8.9 MB.

In the case of Sipht, the improved algorithm reduced the makespan by 5% compared to the original DEWE v3 algorithm, as shown in Figure 3.24. This pattern repeated (with almost significant decreases), the same as for LIGO. In Sipht, few jobs have a single predecessor job that they can schedule with their successor simultaneously. Furthermore, Sipht's average data size is 5.91 MB.

DOI: 10.14750/ME.2024.035

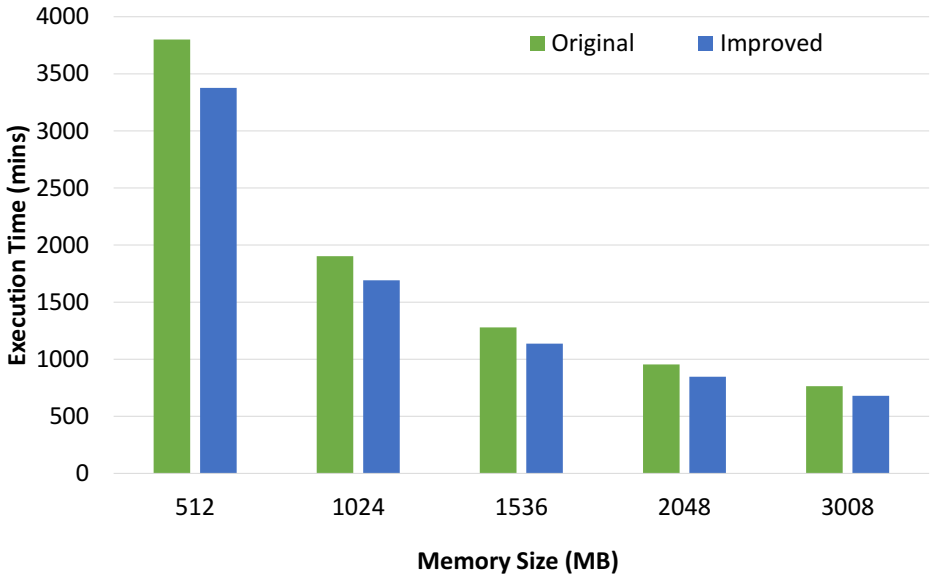


Figure 3.23: *The makespan of the two algorithms with Inspiral (LIGO) workflow running on different Lambda memory sizes.*

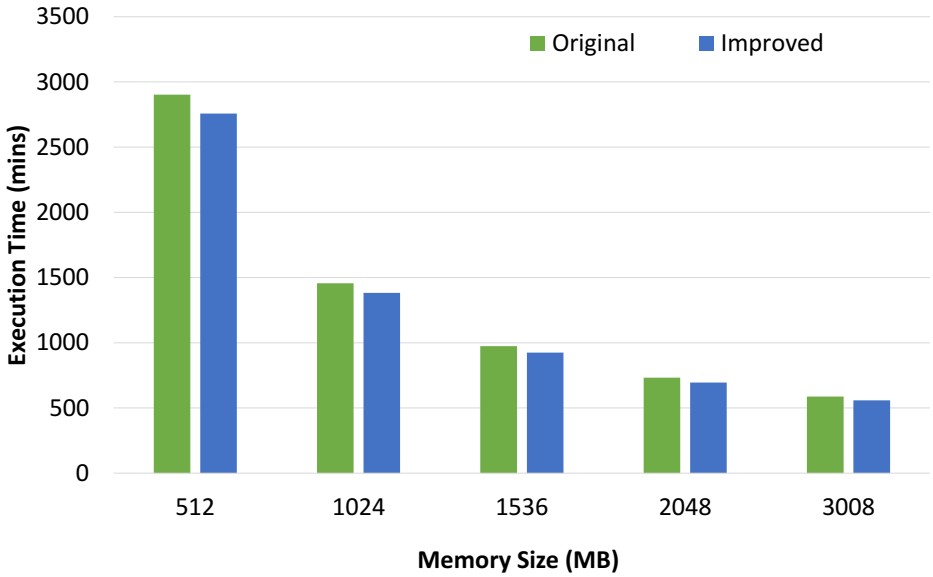


Figure 3.24: *The makespan of the two algorithms with Sipht workflow running on different Lambda memory sizes.*

Finally, the improved algorithm outperforms the original DEWE v3 algorithm in all scientific workflows. The improved algorithm has achieved the best reduction with CyberShake and the least with Sipht. It has achieved a significant reduction with Montage, whose average data size is 4 MB. The most important factor for the improved algorithm is the workflow structure, which ensures that the improved algo-

DOI: 10.14750/ME.2024.035

rithm schedules jobs with precedence requirements into the same invocation. These results are consistent with what was expected for Montage's improved algorithm in Section 3.4.5.1.

3.5 Summary

A scientific workflow application consists of many dependent jobs with complex priority constraints between them. Cloud workflow simulators do not provide sufficient support for the underlying virtualised infrastructure. This includes physical machine state scheduling, virtual machine creation details and virtual machine placement. Other simulators are often user-centric and treat the cloud as a black box. Unfortunately, this behaviour prevents assessing the impact on the infrastructure of the various decisions made by the WMS.

This chapter presented DISSECT-CF-WMS, a workflow management system simulation built on DISSECT-CF. We developed DISSECT-CF-WMS to focus on the user side of the clouds, while DISSECT-CF focuses on the internal behaviour of the IaaS systems. DISSECT-CF-WMS enables better energy awareness by allowing the investigation of physical machine schedulers and customisable consumption characteristics. It also provides dynamic provisioning to meet the resource needs of the workflow application as it runs on the infrastructure, taking into account the provisioning delay of a VM in the cloud. It also provides a serverless simulation for executing scientific workflows on Lambda.

We evaluated our simulator by running several workflow applications with different schedulers of physical machines for a given infrastructure. The experimental results show that workflow researchers can investigate different PM schedulers on infrastructure configurations to achieve lower energy consumption.

The experiments also show that DISSECT-CF-WMS is up to 295$\times$ faster than WorkflowSim and still delivers accurate results. The experimental results of the autoscaling mechanism show that the integration has the potential to optimise makespan, energy consumption, and VM utilisation over static provisioning. This work also allowed us to investigate Internal IaaS behavioural knowledge, such as different scheduling strategies for physical machines in a simulated environment; DISSECT-CF-WMS proved very useful. The experimental results of serverless simulation validated our real-life experiments from the previous chapter.

Workflow scheduling is an increasingly important area for WMS. Therefore, the next chapter will focus on developing a scheduling algorithm using our DISSECT-CF-WMS extension. The developed algorithm can provide further evidence for the benefit of our extension for optimising workflow techniques.

DOI: 10.14750/ME.2024.035

Chapter 4

Structure-Aware Scheduling Algorithm for Deadline-Constrained Scientific Workflows in the Cloud

4.1 Introduction

In the previous chapter, we developed the DISSECT-CF-WMS extension to capture the internal details of cloud infrastructures when executing workflows. We discussed the advantages of using our extension over other workflow simulators. Furthermore, one of the goals of DISSECT-CF-WMS is to enable workflow techniques (e.g., scheduling algorithms). In the previous chapter, We implemented three popular scheduling algorithms (e.g., HEFT, MinMin, and MaxMin). Workflow scheduling is an increasingly important area for WMS. Therefore, this chapter focuses on developing a scheduling algorithm using our DISSECT-CF-WMS extension. The developed algorithm can provide further evidence for the benefit of our extension for optimising workflow techniques.

The objective of the workflow scheduling problem in the cloud is to map tasks to resources to maintain task precedence while achieving certain performance metrics [62]. In the cloud, faster and more powerful computing resources are often more expensive than slower ones. On the other hand, using powerful computing resources can shorten workflow execution time. Consequently, the trade-off between time and cost is a major challenge for cloud-based workflow scheduling [25]. Two typical approaches are used to solve this: reducing the total execution time under a budget constraint [51] and reducing the financial cost under a time constraint [24]. This dissertation presents an approach to the problem of time-constrained workflow scheduling. The objective is to develop a workflow schedule for a given workflow that reduces the monetary cost of running the workflow in the cloud within a given time limit.

DOI: 10.14750/ME.2024.035

Creating an optimal schedule in a heterogeneous cloud environment is NP-hard [69]. On the other hand, workflow scheduling aims to reduce the overall time. Consequently, no algorithm can achieve an ideal solution in polynomial time, while certain algorithms can provide approximate results in polynomial time. Therefore, heuristics are required to find near-optimal solutions effectively.

In a cloud computing environment, it is challenging to select the type and amount of resources to use for the cost-effective execution of scientific workflows [62]. A shorter execution time can be achieved using many resources, but this could come at a significant financial cost. In recent years, a significant amount of research has been conducted on algorithms for scheduling scientific workflows, which are essential for maximising the benefits of cloud computing. However, these algorithms must focus not only on assigning tasks to resources but also on determining the amount and type of resources to be used (i.e., provisioning resources) during the execution of the workflow [61]. Moreover, it is necessary to determine when these resources should be provisioned and when they should be de-provisioned during the workflow execution.

This chapter presents a Deadline and Structure-Aware Workflow Scheduler (DSAWS), which is a heuristic. The algorithm is a static assignment of tasks to VMs with an elastic VM pool that provisions and de-provisions VMs for scheduling tasks as the workflow executes. The algorithm analyses the workflow structure to determine the type and amount of VMs to deploy and when to provision and de-provision them. The algorithm's first phase (the planning phase) selects the number and type of VMs to be used and the allocation of tasks to these resources. In the second phase, the algorithm provisions the VMs selected in the planning phase at the specified times. It also releases these VMs based on the times set in the first phase, considering the delay in provisioning/de-provisioning a VM in the cloud. Its main objective is to use these resources effectively to keep costs down without compromising deadlines.

We evaluated our algorithm using well-known workflows such as Montage, CyberShake, Inspiral, and Epigenomics, as this makes our results comparable to future studies. Finally, the experimental results of the DSAWS algorithm are compared with different scheduling algorithms such as Dyna[73] and CGA[44].

This approach reduces the overall execution cost of a workflow while meeting a user-defined deadline. Experimental results show that DSAWS outperforms other state-of-the-art algorithms in terms of meeting workflow deadlines while reducing execution costs. The experiments have shown that DSAWS delivers more cost-efficient schedules for various workflow applications than Dyna and CGA.

Structure of the chapter: Section 4.2 summarizes the related works and briefly presents previous results. The details of the design and implementation of the DSAWS algorithm are described in section 4.3. The experiment results are shown and discussed in section 4.4. Section 4.5 concludes the chapter.

DOI: 10.14750/ME.2024.035

4.2 Related Works

Many scheduling algorithms have focused on reducing the execution time of workflow applications in cloud computing. Heuristics and meta-heuristics-based approaches have been studied for the workflow scheduling problem.

Genetic algorithms (GA) [72] and Particle Swarm Optimisation (PSO) [58] are well-known meta-heuristic techniques. Moreover, meta-heuristic techniques such as GA and PSO can be found in the literature for workflow scheduling in the cloud. Verma et al.[70] presented a genetic algorithm that schedules cloud-based workflows depending on their importance to reduce the execution cost while meeting the workflow deadline. However, this algorithm does not consider the virtual machines' start-up time in the cloud. The paper [44] presents a genetic algorithm for deadline-constrained scheduling of workflows using the co-evolution technique to modify crossover and mutation probabilities to accelerate convergence and prevent prematurity. These approaches have the potential to be implemented in a cloud environment, although the waiting time might require the use of a computationally intensive meta-heuristic optimisation technique. The pre-processing duration may increase as the workflow size increases, leading to significant queuing delays.

Several heuristic algorithms [2, 18, 29, 38, 71] in the cloud computing environment are presented for workflow scheduling. Saeid et al.[2] presented a deadline-constrained approach for scheduling workflows that allocates an entire critical path to a single VM instance to reduce data transfer time between successive jobs. This technique does not consider allocating jobs from a single path to many VM instances in search of better scheduling options. This technique also does not consider the time it takes a provisioned VM instance to send all output data to the local storage of the VMs running the child tasks before it is de-provisioned. Therefore, this is not practical during the period of process execution.

Xiumin et al.[74] have proposed a technique for extending HEFT [67]. It uses a two-step approach to reduce workflow makespan and execution costs simultaneously. However, it does not consider the startup time of a VM instance or the actual data transfer time between successive jobs. This algorithm selects the final scheduling solution from the K best solutions. However, the optimal determination of the value of K is not addressed. In the meantime, comparing the K scheduling solutions to choose the best one results in the scheduling algorithm's inefficiency.

The Coevolutionary Genetic Algorithm (CGA) [44] was proposed based on the biological evolutionary method (genetic algorithm), where the adaptive penalty function for strict deadlines was introduced. It assigns partial deadlines to each task and executes them on currently rented or existing VMs to reduce the total cost. CGA's static approach has the potential to generate optimal solutions. However, its disadvantage is its inability to meet deadlines when unexpected delays occur.

DOI: 10.14750/ME.2024.035

Dyna [73] is a scheduling technique developed with auto-scaling capabilities for the cloud to dynamically provision and de-provision VMs depending on the current state of tasks. It was presented to develop a scheduling system that reduces the expected monetary cost under user-defined probabilistic scheduling constraints. It selects VM types for each workflow task to reduce costs based on an A-star search. It is designed to schedule many workflows simultaneously but can also be modified to schedule only one. Dyna is periodically improved by adjusting the number of VMs requested in each category to ensure the timely completion of tasks at a lower cost.

ARPS [59] is an algorithm for adaptive resource provisioning and scheduling for scientific workflows in Infrastructure as a Service (IaaS) clouds. It was designed to address cloud-specific issues such as unlimited on-demand access, heterogeneity, and pay-per-use (i.e., per-minute billing). Consequently, their strategy was also designed to consider a user's deadline and reduce the cost of the environment by using the resource provisioning and scheduling service. Finally, their experimental results show that they perform a workflow more effectively than other sophisticated algorithms to meet deadlines and reduce costs.

Mao et al.[48] proposed a workflow scheduling heuristic for the cloud environment that allows them to dynamically generate the lowest schedule while meeting the user's deadline. They investigated multiple VM types and cloud characteristics, such as alternative pricing models and acquisition delays. However, they did not consider data transfer time between linked jobs, which is one of the most important criteria and significantly impacts data-intensive workflows.

By analysing the workflow structure, [37] proposes a resource provisioning and scheduling technique that determines the required number and configuration of VMs. They claimed that their approach addresses data-intensive workflows to minimise data transfer. However, they did not consider the data transfer time between tasks during the execution of the two examples presented, which is one of the most important factors and significantly impacts workflow execution time. In addition, they neglected resource provisioning and de-provisioning delays in their experiments.

Researchers in [42] have presented a two-step method for provisioning cloud resources for workflows by minimising makespan and wastage of resources based on their structural characteristics. The proposed method considers the nature of the tasks, which may be computational, memory-, or storage-intensive. The performance of the presented algorithm is evaluated using five scientific workflows as benchmarks. Simulation results show that the proposed method outperforms two existing algorithms for each workflow.

Although there are several workflow scheduling techniques, there is a need for resource estimation for workflow execution because the above approaches have not analysed the workflow structure in depth. In this chapter, we propose DSAWS, which is a complete full-ahead scheduling algorithm that considers the structure of the

DOI: 10.14750/ME.2024.035

workflow. We discuss a method to deal with under- and over-provisioning issues.

The comparison of the discussed scheduling algorithms can be seen in Table 4.1. For the comparison, we listed the scheduling type and the scheduling objectives, and we also indicated the evaluation if they have considered simulation or real-world experiments.

Table 4.1: *Comparison of algorithms for the scheduling model.*

Algorithm	Scheduling Type	Scheduling Objectives	Evaluation
Saeid et al. [2]	Static	Deadline & Cost	Simulation with synthetic workflows of arbitrary size
Xiumin et al. [74]	Static	Makespan & Cost	Simulation with real-world and synthetic workflows
CGA [44]	Static	Deadline & Cost	Simulation on WorkflowSim with 4 scientific workflows
Dyna [73]	Dynamic		Amazon EC2 with 3 scientific workflows
ARPS [59]	Static & Dynamic	Deadline & Cost	Simulations on CloudSim with 4 scientific workflows
Mao et al.[48]	Dynamic	Deadline & Cost	Simulation with Pipeline, Parallel and Hybrid applications
Kanagaraj et al. [37]	Static	Makespan & Resources utilization	Simulation with data-intensive workflows

4.3 The Proposed Scheduling Algorithm

Several objectives associated with task scheduling issues need to be addressed. The approach suggested in this chapter focuses on running workflow applications in a cloud environment to lower overall execution costs while still meeting the user-set deadline. The proposed technique analyses the workflow structure, determines the number of tasks at each level, and provides a rank value for all workflow tasks. To determine the quantity and configuration of resources needed to complete the workflow execution by the user-set deadline, use this rank value.

DOI: 10.14750/ME.2024.035

In this chapter, two approaches are discussed. First, in the planning phase, the exact number and configuration of VMs that need to be rented from cloud service providers are determined based on the deadline constraint and the ranking value of the tasks. It also uses the remaining time (leftover time) in the current billing period to avoid wasting resources. The plan to reuse cloud resources can eliminate the need for further provisioning and deployment costs.

The second approach concerns the execution phase (the second phase). It aims to provision or de-provision the resources of the selected services for tasks in the planning phase. These resources are maintained until they have completed all the previously assigned tasks. However, if some resources are not needed for the subsequent tasks, they are terminated immediately after the output data is transferred. This significantly reduces execution time and resource costs, which is crucial for workflow users. We will explain the steps of Algorithms 1 and 2 in the next paragraph using Table 4.2, which contains the notations used in our algorithms.

Algorithm 1 calculates the rank value of each task, starting with the exit tasks (tasks without any child). First, the runtime of each exit task became its rank value for those tasks that have no child tasks (lines 2-6), and then the rank value is assigned to the parent tasks of the exit tasks (lines 7-15), which involves calling Algorithm 2 (line 11).

Second, Algorithm 2 assigns to each parent task the maximum rank value of the rank values of its child tasks (lines 2-8) with the maximum data size of the data sizes of its child tasks (lines 9-12). Algorithm 2 continues assigning the rank value for each task recursively until it reaches the entry tasks that have no parent tasks (lines 15-19). Finally, after Algorithm 2 completes its steps, Algorithm 1 sorts all tasks in descending order according to their rank values to determine the order in which workflow tasks should be scheduled (line 16). In the next paragraphs, we will explain the steps of the Algorithms 3 and 4.

The pseudocode of the entire DSAWS algorithm for workflow scheduling is shown in Algorithm 3. The proposed algorithm uses the rank value to support each task by selecting the appropriate VM to execute it within the deadline. In the first phase, the algorithm selects the appropriate type and the exact number of VMs needed to execute workflow tasks to meet the deadline set by the user. After the basic initialisation in lines 2-8 of Algorithm 3, it receives the workflow tasks arranged from Algorithm 1 while the deadline D is set by the user. Line 2 identifies the available instance types of VMs the service provider offers. In line 3, the rented set $rentedVMs$ is empty at the beginning of the execution of the algorithm. We have initialised a variable called $success$ that changes when a task finds its matching VM to meet the deadline. In line 6, $vm_{minTime}$ is the earliest available VM time in the currently leased VMs. In line 7, although all tasks are arranged in descending order of their rank values, Algorithm 3 selects ready tasks from the $rankList$ and adds them periodically to the $readyList$ in

DOI: 10.14750/ME.2024.035

Table 4.2: *Notations for the symbols used in the algorithms.*

Notations	Meanings
$T(G)$	Set of tasks in workflow graph.
D	User-defined deadline of the workflow.
E	Set of edges between tasks in the workflow.
t_{entry}	Task without any parent.
t_{exit}	Task without any child.
t_{EST}	Earliest Start Time of task t.
t_p	Predecessors (parents) of task t.
p_p	Predecessors of predecessor p.
t_{ch}	Successors (children) of task t.
$t_{runtime}$	Runtime of task t.
t_{rank}	Rank value of task t.
$ch_{maxRank}$	Maximum rank value of child task ch.
$ch_{maxData}$	Maximum data size of child task ch.
ch_{rank}	Rank value of child task ch.
ch_{data}	Data size of child task ch.
$p_{runtime}$	Runtime of parent task p.
p_{rank}	Rank value of parent task p.
S	Set of available instance types of VMs the service provider offers.
$rentedVMs$	Set of virtual machines currently rented by the algorithm.
$vm_{booting}$	Booting time of virtual machine vm.
$vm_{shutdown}$	Shutdown time of virtual machine vm.
$vm_{minTime}$	the earliest available time of vm in VM.
$readyList$	List of the ready tasks in the workflow.
$rankList$	List all tasks in descending order of their rank values.
$timeLine$	The difference of subtracting $vm_{minTime}$ or t_{EST} from D.
s^{speed}	Performance capacity of service type s.
vm^{speed}	Performance capacity of virtual machine vm.
$VMsList$	List of selected VMs with scheduled tasks on them during the planning phase.
m	Number of VM types.
n	Number of currently leased VMs.
vm_{start}	Start time of virtual machine vm.
vm_{stop}	Stop time of virtual machine vm.
$vm_{idleTime}$	Idle time of virtual machine vm.
$vm_{billingPeriod}$	Billing period of virtual machine vm.
$t_{transferTime}$	Transfer time of all output data of task t to the VMs of its successors ch.
$vm_{idleTime}$	Calculated idle time between two consecutive tasks on virtual machine vm.
t_{vm}^{start}	Start time of task t on virtual machine vm.
t_{vm}^{end}	End time of task t on virtual machine vm.

DOI: 10.14750/ME.2024.035

Algorithm 1 Workflow Ranking

```
1: procedure AssignRanking(T(G))
2:     for all t ∈ T(G) do
3:         if t has no children then
4:             t_rank := t_runtime
5:         end if
6:     end for
7:     for all t ∈ T(G) do
8:         if t has no children then
9:             for each parent p of t do
10:                if p has no rank value then
11:                    call TaskRank(p)
12:                end if
13:            end for
14:        end if
15:    end for
16:    Arrange all tasks in the list rankList in decreasing order of rank values.
17: end procedure
```

Algorithm 2 Task Ranking

```
1: procedure TaskRank(p)
2:     ch_maxRank := 0
3:     ch_maxData := 0
4:     for each child ch of p do
5:         if ch has rank value then
6:             if ch_rank > ch_maxRank then
7:                 ch_maxRank := ch_rank
8:             end if
9:             if ch_data > ch_maxData then
10:                ch_maxData := ch_data
11:            end if
12:        end if
13:    end for
14:    p_rank := p_runtime + maxRank + maxData
15:    if p has parent then
16:        for each parent p_p of p do
17:            call TaskRank(p_p)
18:        end for
19:    end if
20: end procedure
```

DOI: 10.14750/ME.2024.035

order. In line 8, $timeLine$ is the difference between the earliest available time of the VM or the earliest start time of a task and a deadline D. The while loop in line 9 is used to find a suitable VM for each task in the workflow. In line 12, the $timeLine$ is the difference resulting from subtracting $vm_{minTime}$ from the deadline because the task begins its execution by selecting a VM instance that has already been rented. First, the ready tasks check the available rented VMs to meet the deadline. If a task does not find a suitable VM to meet the deadline, it selects a new suitable VM to meet the deadline. At the beginning of the execution of the algorithm, there are no rented VMs in line 13. Therefore, the algorithm skips lines 13-20. In line 22, the $timeLine$ is the difference resulting from subtracting the earliest start time of a task (t_{EST}) from the deadline since the task begins its execution by selecting a new VM instance. Line 23 tries to select a new VM by comparing $timeLine$ with the task's rank value divided by the VM speed (lines 13 and 23). For cost-effective task scheduling, the task searches for a VM at the service provider, starting with the slowest VM until it reaches the appropriate VM that meets the deadline (lines 24-25). In line 26, the task is removed from the unscheduled $readylist$, while in line 28, the selected new VM is added to the set of rented VMs ($rentedVMs$). The algorithm updates the EST for all successors of a task (line 16 or 27) after finding a suitable resource in line 15 or 25. This update may change the readiness of the tasks based on the completion time of their predecessor tasks. When all tasks are assigned to VMs, the algorithm calls algorithm 4 in line 33.

Algorithm 4 shows the pseudocode of the TimelineVMS algorithm for provisioning and de-provisioning resources. In the second phase, the algorithm first determines the time for provisioning the VMs and the time at which each VM is de-provisioned by taking into account the delays in provisioning and de-provisioning a VM in the cloud. Second, the algorithm determines the idle time between two scheduled, consecutive tasks on each VM. During the execution of the workflow, the algorithm dynamically adds and removes resources from its pool.

Algorithm 4 represents the second phase, where workflow tasks are scheduled on the selected resources ($VMsList$) during the planning phase. It receives from Algorithm 3 a schedule for all tasks about the types and number of their VMs ($VMsList$). After initialisation in lines 2-5, the booting and shutdown times of resources and the VM's billing period are set. In line 5 of the algorithm, $vm_{idleTime}$ is used to find the idle time between any two scheduled consecutive tasks on a VM to shut down this VM.

To do this, the VM's billing period is taken into account to determine whether the idle time is greater than the billing period of a VM. For example, if workflow tasks are scheduled on VMs in the first phase, the algorithm determines when to start a VM and when to shut it down in the second phase by checking the schedule of the tasks on their VMs. This reduces the idle time of VMs. In lines 6 and 7, the algorithm

DOI: 10.14750/ME.2024.035

identifies the tasks of each VM by reading the start and end times of each task on it. The algorithm then attempts to prepare tasks' resources before the tasks begin their execution (lines 9-12), as the provisioning process is still significant due to the overhead associated with leasing virtual machines (lines 8–14). The consequences of VM provisioning and de-provisioning delays are greatly mitigated and are much easier to manage.

First, the algorithm uses resource elasticity to meet the user's deadline but knows when to rent and release resources. If a new VM needs to be provisioned during the execution of the workflow, the algorithm can start VMs earlier before the task starts by taking into account the delay in provisioning a VM instance to speed up the execution of the workflow because provisioning a VM takes time. Secondly, it uses the cloud billing model to optimise resource utilisation while reducing the number of rented resources. It also tries to schedule tasks on currently rented VMs to reduce the need for further VM provisioning costs.

Furthermore, the algorithm checks the timeline of each VM to see if the idle time is greater than the instance's billing period (lines 16-20). It then sends the output data to the VMs performing the successor tasks (line 17) before de-provisioning that VM instance in line 19. Finally, it sends the output data to the VMs executing the successor tasks, if any (line 22), before de-provisioning that VM instance in line 24 because the VM has completed its tasks.

4.3.1 An illustrative example

To illustrate how the proposed algorithm works, we apply its steps to a sample workflow shown in Figure 4.1. The workflow consists of nine tasks in the nodes of the graph: $t_1 - t_9$. The value within the node of each task indicates the estimated time of its execution (in seconds), while the number in parentheses represents the rank value. The estimated time for data transfer between VMs is also shown on the edges between nodes.

The following sections explain how to use the new algorithm to perform the workflow. Before the algorithm starts, the rank value for all tasks should be calculated using Algorithms 1 and 2. Then, the tasks are sorted in descending order of their rank value. We assume that the cloud provider offers three types of VM computing services (vm^1, vm^2 and vm^4) to execute the workflow tasks. The billing period for computing services is set to 10 seconds, and the costs for vm^1, vm^2 and vm^4 are 2, 4 and 6 respectively. The speeds for vm^1, vm^2 and vm^4 are 1, 2 and 4, respectively. The VM instance provisioning and shutdown delays are set to 2 and 1 second, respectively, and the workflow deadline is set to 35, which is the maximum rank value (32) in the workflow, plus the provisioning (2) and de-provisioning delays (1).

For the example workflow in Figure 4.1, we call the DSAWS scheduling algorithm,

DOI: 10.14750/ME.2024.035

Algorithm 3 The DSAWS scheduling algorithm

1: **procedure** DSAWS(G(T,E),D)
2: m= available instance types of VMs (S)
3: $rentedVMs = \varnothing$ the currently leased virtual machines
4: $success$ = false.
5: $vm_{booting}$ = the booting time of VM
6: $vm_{minTime}$ = the earliest available time of vm in $rentedVMs$.
7: $readyList$ = receives repeatedly ready tasks from $rankList$.
8: $timeLine$ = represents the difference of subtracting $vm_{minTime}$ or t_{EST} from the deadline D.
9: **while** (there exists unscheduled t in $readyList$) **do**
10: t = find the earliest EST in $readyList$
11: $vm_{minTime}$= find the earliest available time of vm in $rentedVMs$.
12: $timeLine$:= D - $vm_{minTime}$
13: **for all** $vm_j \in VM$ **do** where $j = 1, 2, \ldots, n$
14: **if** $timeLine >= \frac{t_{rank}}{vm_j^{speed}}$ **then**
15: select vm_j^{speed} to run t
16: update EST for all successors of t
17: remove t from $readyList$
18: $success$:= true
19: **end if**
20: **end for**
21: **if** success==false **then**
22: $timeLine$:= D - t_{EST}
23: **for all** $s_i \in S$ **do** where $i = 1, 2, \ldots, m$
24: **if** $timeLine >= (\frac{t_{rank}}{s_i^{speed}})$ **then**
25: select a new instance vm_i^{speed} to run t
26: remove t from $readyList$
27: update EST for all successors of t
28: add vm_i^{speed} to $rentedVMs$
29: **end if**
30: **end for**
31: **end if**
32: **end while**
33: call TimelineVMs(VMs)
34: **end procedure**

i.e., Algorithm 3. At the beginning of the workflow execution, $t_1 - t_3$ are the ready tasks that need to be scheduled and steps 13-20 of Algorithm 3 are not applied since no VMs have been provisioned yet. Therefore, steps 21-31 are executed, running the for loop in line 23 one or more times until each task finds its appropriate resource (s_1^1) to execute that meets the user's deadline. The value EST is calculated for the

DOI: 10.14750/ME.2024.035

Algorithm 4 Provisioning resources

```
1: procedure TIMELINEVMS(VMsList)
2:     vm_booting = the booting time of VM
3:     vm_shutdown = the de-provisioning time of VM
4:     vm_billingPeriod = the billing period for VM
5:     vm_idleTime = the idle time between two consecutive tasks on the VM.
6:     for all vm ∈ VMsList do
7:         for each task t on vm do
8:             if vm has not provisioned then
9:                 vm_start=(t_start − vm_booting)
10:                if vm_start < 0 then
11:                    vm_start=0
12:                end if
13:                provision vm on the time of vm_start
14:            end if
15:            vm_idleTime= vm_idleTime - vm_shutdown
16:            if vm_idleTime >= vm_billingPeriod then
17:                transfer output data of t to the VMs of its successors.
18:                vm_stop= t_end+t_trasferTime
19:                de-provision vm on the time vm_stop
20:            end if
21:        end for
22:        transfer output data of t to the VMs of its successors.
23:        vm_stop= t_end+t_trasferTime
24:        de-provision vm on the time vm_stop
25:    end for
26: end procedure
```

successor tasks of t_2 in step 25.

Steps 13-20 can be executed if some resources are available. A task checks the available rented resources (vm_1^1), starting with the slowest and then the fastest (in ascending order by speed). If a task (t_1) does not find a suitable resource that completes execution within the deadline, it decides to start a new instance of available services (s_2^1) considering the speed of the resource in step 24. Similarly, t_3 will select a new instance (s_3^1) that can complete execution within the deadline. Table 4.3 shows the scheduled tasks, the selected VMs, and the execution time (in seconds) of each task.

Step 1: First, the DSAWS algorithm assigned t_2, t_1, and t_3 to vm_1^1, vm_2^1, and vm_3^1, respectively. The algorithm started three VMs to meet the user's deadline, and the current simulation time was two due to the VM booting time.

Step 2: The algorithm assigned t_5 to the available instance vm_1^1, so no data transfer occurred. The same is occurred for steps 3 and 4: t_4 and t_6 were assigned to

DOI: 10.14750/ME.2024.035

the instances of their predecessor tasks vm_2^1 and vm_3^1, respectively. Finally, the last three steps used the same available instances of their predecessor tasks without data transfer. After all, tasks have been scheduled. The next step is to invoke Algorithm 4 in step 33 to provision and de-provision the resources of the services assigned to the tasks during the previous phase (the planning phase).

Finally, Algorithm 4 receives from Algorithm 3 the schedule (e.g., Table 4.3) indicating the time of execution of each workflow task on each resource of the service type. In lines 2-5, the algorithm sets several variables, e.g., the periods for starting up (e.g., 2) and shutting down (e.g., 1) of the resource. The variable in line 4 is the instance's billing period (e.g., 10). In line 5, this variable will check the idle time between any two consecutive tasks on each VM. The for loop in line 6 is executed for all VMs assigned during the planning phase ($vm_1^1 - vm_3^1$). Then, the for loop in line 7 is executed for all tasks on each VM (e.g., t_2, t_5 and t_8 on vm_1^1). Since no VM is provisioned at the beginning of the workflow execution, the delay in booting the VM cannot be avoided (lines 8-14).

However, the other tasks ($t_4 - t_9$) that start at a time greater than the booting delay can start executing and thus avoid the VM booting delay. The algorithm provisions VMs (vm_{start}) in advance of the tasks' start times (line 9), taking into account the VM provisioning delay ($vm_{billingPeriod}$). Finally, if a VM has subtracted the shutdown delay time from the VM idle time (line 16) and the difference is greater than or equal to the instance's billing period (line 16), the VM is terminated immediately after the output data is transferred to the VMs of its successors (lines 15-19).

Furthermore, if no more tasks are running on a VM, the VM is also terminated immediately after the output data has been transferred to the VMs of its successors (lines 22-24). The makespan for the workflow with the selected VMs ($vm_1^1 - vm_3^1$) is 30 seconds. Taking into account the data transfer time and the delay times for booting and shutting down the VM instances, the total cost of the sample workflow is 18.

4.4 Evaluation

Our experiment evaluated DSAWS with other competitive algorithms like CGA as a static algorithm and Dyna as a dynamic algorithm. CGA was chosen for comparison in our evaluation because of its static approach, which has the potential to generate optimal solutions. Dyna was chosen for comparison in our evaluation because the algorithm is periodically improved by adjusting the number of VMs requested in each category to ensure the timely completion of tasks at a lower cost. The aim is to show how the static component of DSAWS enables the creation of schedules that outperform the Dyna algorithm in terms of meeting workflow deadlines while reducing execution costs.

DOI: 10.14750/ME.2024.035

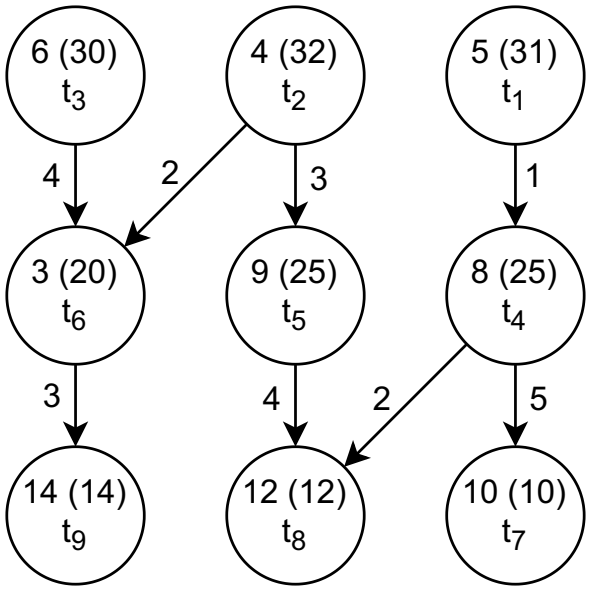


Figure 4.1: *A sample workflow.*

Table 4.3: *The scheduling of the workflow tasks for each step of executing DSAWS on the sample workflow of Figure 4.1*

Step	Task	Rank	Current Sim Time	timeLine	$\frac{t_{rank}}{vm_j^{speed}}$	VM selection	Start	End	VM cycle
1	t_2	32	2	32	32	vm_1^1	2	6	1
1	t_1	31	2	32	31	vm_2^1	2	7	1
1	t_3	30	2	32	30	vm_3^1	2	8	1
2	t_5	25	6	28	25	vm_1^1	6	15	2
3	t_4	25	7	27	25	vm_2^1	7	15	2
4	t_6	20	8	26	20	vm_3^1	8	13	2
5	t_9	14	13	21	14	vm_3^1	13	27	3
6	t_8	12	15	19	12	vm_1^1	15	29	3
6	t_7	10	15	19	10	vm_2^1	15	25	3

The experiment was conducted in the DISSECT-CF-WMS [4] simulator, which is an extension of the DISSECT-CF simulator. It is useful for running scientific workflows on cloud resources. DISSECT-CF-WMS focuses on the user-side behaviour of clouds, while DISSECT-CF focuses on the internal behaviour of IaaS systems. It also supports dynamic provisioning to meet the resource requirements of the workflow application while running on the infrastructure, taking into account the provisioning and de-provisioning delays of a cloud-based VM.

DOI: 10.14750/ME.2024.035

We analysed the most widely used workflows to demonstrate the importance of the DSAWS algorithm. We chose the well-known workflows Montage from the field of astronomy, CyberShake from the field of physics, Inspiral (LIGO) from the field of astrophysics and Epigenomics from the field of bioinformatics. Workflows with about 1,000 tasks were used for the evaluation. All relevant characteristic values required for the above algorithms are listed in Table 4.4 for the analysis of experiments. The performance of the four workflows in DSWAS is compared with the Dyna and CGA approaches.

Table 4.4: *The characteristics values for each workflow application*

Workflow type	Number of levels	Number of tasks	Number of dependencies	Mean run-time (sec.)	Mean data size (MB)
Montage	9	1000	4485	11.37	3.21
CyberShake	5	1000	3988	22.75	102.29
LIGO	6	1000	3246	227.7	8.9
Epigenomics	8	997	3228	3866.4	388.59

We created a model of the cloud infrastructure of Google Cloud Engine[1] with different VM configurations selected from the predefined machine types of the cloud. An IaaS provider with a single data region and seven types of VMs was set up. Table 4.5 shows the VM setup type based on Google Compute Engine offerings. For Google Cloud Engine, the core of Compute Engine CPU provides a minimum processing capacity of 2.75 GCEUs (2.75 ECUs), or about 2750 MIPS [3]. A billing slot of 60 seconds was modelled, as service providers such as Google Compute Engine and Microsoft Azure offered. Provisioning delay was set to 30 seconds [63] and deprovisioning delay to 3 seconds [49] for all types of VMs. The bandwidth between VMs was set to 1 Gbit.

Table 4.5: *Types of VM based on Google Compute Engine offering*

Name	Memory (GB)	Google compute engine units	Price per minute ($)
n1-standard-1	3.75	2.75	0.00105
n1-standard-2	7.5	5.5	0.0021
n1-standard-4	15	11	0.0042
n1-standard-8	30	22	0.0084
n1-standard-16	60	44	0.0168
n1-standard-32	120	88	0.0336
n1-standard-64	240	176	0.0672

[1] `https://cloud.google.com/compute/all-pricing`

DOI: 10.14750/ME.2024.035

To evaluate the ability of each approach to achieve a valid solution that meets the deadlines, we set the success rate metric, which is calculated as the proportion of the current execution times to the given deadlines. For the evaluation, we set three deadline factors based on the maximum rank value of each workflow. The maximum rank value represents the strict deadline factor (1), as shown in Table 4.6. In contrast, the moderate and relaxed deadlines are obtained by multiplying the maximum rank values by (1.5) and (2), respectively.

Table 4.6: *The maximum rank values in seconds for each scientific workflow.*

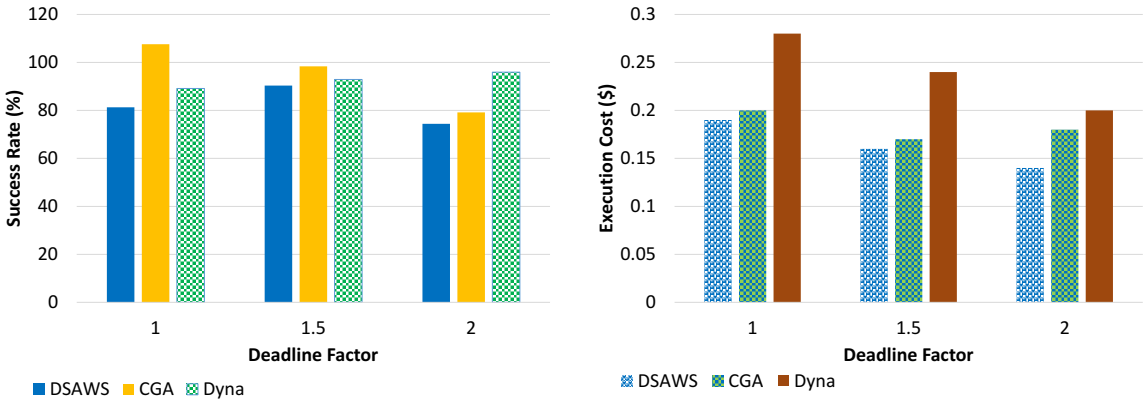
Workflow type	The maximum rank value (strict Deadline factor)
Montage	369 seconds
CyberShake	736 seconds
LIGO	625 seconds
Epigenomics	27232 seconds

Figures 4.2a, 4.3a, 4.4a, and 4.5a show the results of the success ratios for each workflow with the three algorithms. On the other hand, Figures 4.2b, 4.3b, 4.4b, and 4.5b show the execution costs (in $) for each workflow with the same algorithms.

In the case of Montage workflow, DSAWS and Dyna algorithms completed the execution of the workflow within the deadline, while CGA failed to meet the strict deadline factor, as shown in Figure 4.2a. Dyna met all deadline factors, as shown in Figure 4.2a. The DSAWS approach met all deadlines with the lowest cost compared to the other algorithms, as seen in Figure 4.2b. The Montage workflow has many parallel tasks with a short execution time in the second level. This drastically increases the overall cost of the workflow as more resources are consumed by Dyna, as shown in Figure 4.2b. However, DSWAS overcomes this disadvantage by using the leftover time of resources to save costs. Furthermore, Montage has nine levels and six of these levels are controlled by the single-thread jobs with a total execution time of 332 seconds. Levels 3 and 4 have 142 seconds, which is more than two instance cycles, with the billing period being 60 seconds. Levels 6-9 have 190 seconds, which is equivalent to three instance cycles. Therefore, the DSAWS algorithm keeps only one VM during these periods to reduce the execution cost and meet the deadline.

In the case of the CyberShake workflow, which has a data transfer bottleneck for most scheduling algorithms. This drawback is eliminated by the DSAWS described in this chapter, which allocates resources to all tasks based on their rank value. It guarantees that all tasks are completed within the deadline and starts new instances only when needed. Therefore, DSAWS reduces data transfer by assigning tasks to the same set of resources. The CGA scheduler could not meet the deadline for all deadline factors successfully. While Dyna met the relaxed deadline factor, it failed to meet the other deadline factors. DSAWS, on the other hand, meet all deadlines

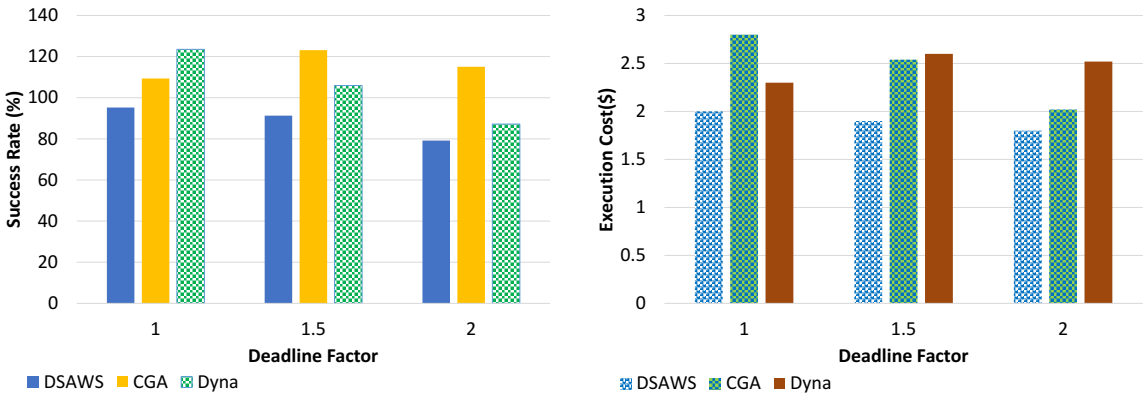
DOI: 10.14750/ME.2024.035



(a) *Makespan* **(b)** *Cost*

Figure 4.2: *The makespan and execution cost of the three algorithms with the Montage application.*

with the lowest execution cost, as shown in Figure 4.3a and Figure 4.3b, respectively. CyberShake has five levels, with most tasks at levels 2 and 3 totalling 994 tasks out of 1000. This results in high concurrency and a large amount of data transfers. CyberShake is a compute- and data-intensive workflow. In addition, level two has 497 tasks with 95.35% of the total execution time of the workflow tasks. As a result, the Dyna and CGA algorithms launched many instances of the computation service, and this has led to an increase in the makespan and execution cost of the workflow due to the increase in data transfers between resources.

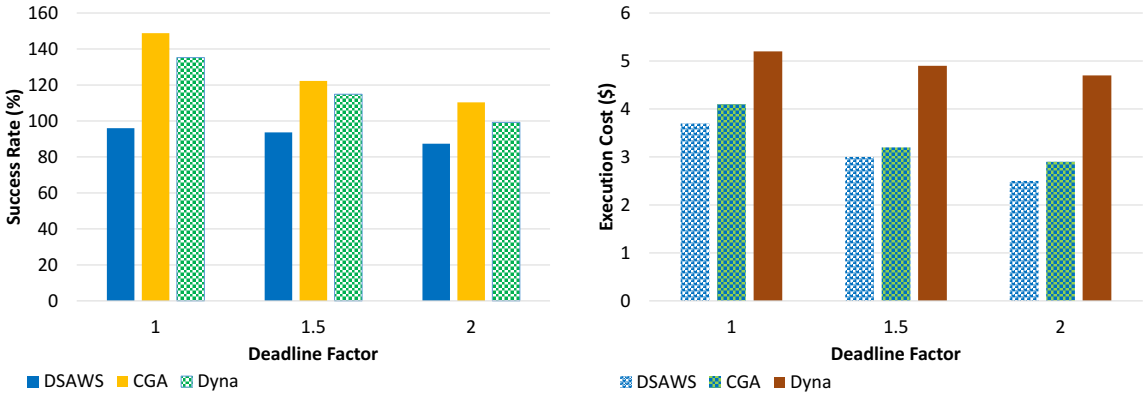


(a) *Makespan* **(b)** *Cost*

Figure 4.3: *The makespan and execution cost of the three algorithms with the CyberShake application.*

DOI: 10.14750/ME.2024.035

In LIGO, DSAWS successfully met all deadline factors, while CGA failed to meet all deadline factors. Dyna met the relaxed deadline factor but failed to meet the other deadline factors, as shown in Figure 4.4a. CGA and Dyna perform badly because fewer resources are available for tasks with long execution times. LIGO is a data and CPU-intensive workflow, and this slowed down the execution of the workflow significantly. However, the proposed technique analyses the workflow structure, determines the number of tasks at each level and provides a rank value for all workflow tasks. The algorithm then assigns the appropriate type of resources to these tasks in the workflow and executes them to meet the user-specified deadline, as shown in Figure 4.4a. Also, unlike the other algorithms, DSAWS achieved the cheapest cost among all schedules, as shown in Figure 4.4b. LIGO has 483 tasks with runtimes greater than the mean execution time (e.g. 227.7). The time difference between tasks can be up to 3 $times$ the mean runtime of the workflow tasks. This results in idle time for other resources.



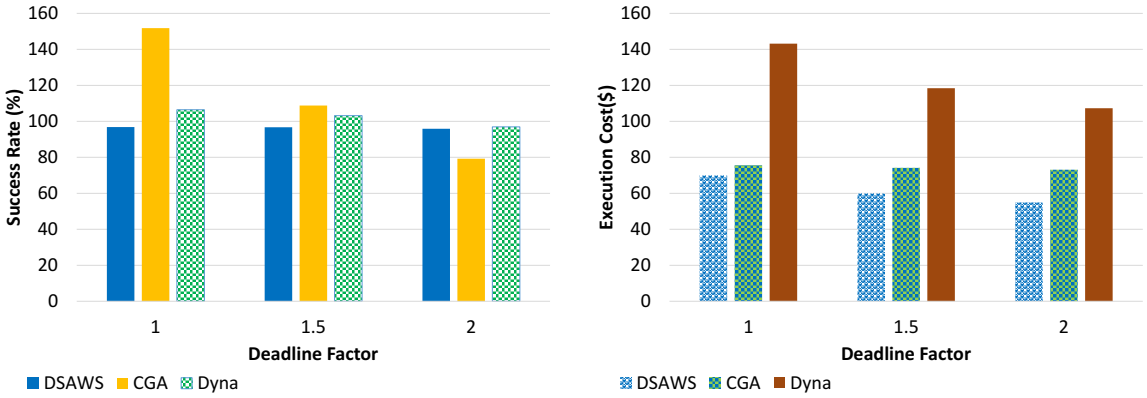
(a) *Makespan* **(b)** *Cost*

Figure 4.4: *The makespan and execution cost of the three algorithms with the LIGO application.*

In the Epigenomics workflow, the CGA scheduler did not successfully meet the deadline for the strict and moderate deadline factors, but it was able to meet the relaxed deadline factor. Similarly, Dyna has met the relaxed deadline factor but failed to meet the moderate and strict deadline factors. For some Epigenomics tasks, there are significant differences in execution times by 15000 $times$ or even more. Therefore, the CPU performance reduction will significantly impact the processing time of these tasks and lead to delays for CGA and Dyna. The DSAWS algorithm, on the other hand, met all deadlines, as shown in 4.5a. Furthermore, unlike the other two algorithms, DSAWS has the lowest execution cost, as shown in Figure 4.5b. This pattern is repeated in Epigenomics experiments, but the time difference can be up to 7 $times$ of the average runtime of the workflow tasks (e.g. 3866.4). Epigenomics has

DOI: 10.14750/ME.2024.035

eight levels, with most tasks at level 5 comprising 245 tasks and 99.8% of the total workflow execution time. These differences show that there is a significant need for resources at this level of the workflow for CGA and Dyna.



(a) *Makespan* **(b)** *Cost*

Figure 4.5: *The makespan and execution cost of the three algorithms with the Epigenomics application.*

Finally, the DSAWS algorithm met all the deadline factors of each workflow, while the CGA and Dyna approaches met 25% and 50% of all the deadline factors of all workflows, respectively. These results are consistent with what was expected for each algorithm. The static heuristic (e.g., CGA) was not more successful in meeting deadlines, but the adaptability of Dyna allows it to meet its aim more frequently. The experiment's results also show the efficiency of DSAWS in terms of its ability to produce more cost-effective schedules. DSAWS outperformed all other algorithms we compared it with in all situations. DSAWS succeeds at the lowest cost compared to CGA and Dyna algorithms. Moreover, CGA showcases its ability to generate more cost-effective schedules and surpasses Dyna by about 92% regardless of whether the deadline was met or not. For some workflow structures (e.g., CyberShake and Epigenomics), our proposed algorithm uses the initial leased VMs to schedule all tasks of the same workflow to minimise data transfer costs. Other structures (e.g., Montage and LIGO) have many tasks with a short execution time, and many instances of the computation service are launched while only a small part of their time interval is used. Therefore, the proposed algorithm uses the remaining time in the current billing period of the VMs to avoid wasting resources. An additional feature of DSAWS evident in the results is its ability to increase the time required to execute the workflow incrementally. The significance of these relationships is that many users are willing to trade off execution time for lower costs, while others are willing to pay higher costs for faster execution. The algorithm must behave within this logic so that the deadline number is perceived as fair by the users.

DOI: 10.14750/ME.2024.035

There are different structures for scientific workflows. In this chapter, the structure of the receiving workflow is analysed to determine the type and number of VMs while meeting the deadline for the workflow. We consider execution time as a form of Quality of Service (QoS) metric. Our goal is to optimise the performance of the workflow in the cloud system by allowing the user to constrain the previously mentioned QoS parameter, namely the deadline. In addition, the user can set optimisation goals such as the minimum makespan. Our timely resource provisioning optimises the execution of workflows within a deadline. Resource selection ensures tasks are completed without delay by comparing rank values with the deadline to select the appropriate VM that meets the deadline.

4.5 Summary

When scheduling workflows in the cloud, resource allocation is important. A good resource estimation method helps the user to reduce the cost and time of workflow execution. Numerous algorithms face the challenge of meeting the user's deadline requirements while minimising the cost of running the workflow. The DSAWS scheduler presented in this chapter analyses the structure of the incoming workflow and assigns an optimal resource provisioning mechanism based on the deadline constraint and the rank values of the tasks in the workflow. The main implementation of this algorithm is to make the second phase follow the schedule of the first phase (scheduling of workflow tasks on selected resources). We evaluate the performance of our algorithm by simulating it with four synthetic workflows based on real scientific workflows with different structures. For some structures (e.g., CyberShake and Epigenomics), our proposed algorithm uses the initial leased VMs to schedule all tasks of the same workflow to minimise data transfer costs. Other structures (e.g., Montage and LIGO) have many tasks with a short execution time, and many instances of the computation service are launched while only a small part of their time interval is used. Therefore, the proposed algorithm uses the remaining time in the current billing period of the VMs to avoid wasting resources. The proposed algorithm reduces the overall execution cost of a workflow while achieving a deadline set by the user. Experimental results show that DSAWS outperforms the Dyna and CGA algorithms in terms of meeting workflow deadlines while reducing execution costs. DSAWS met all the deadline factors of each workflow, while CGA and Dyna met 25% and 50%, respectively, of all the deadline factors of all workflows.

DOI: 10.14750/ME.2024.035

Chapter 5

Conclusion

This dissertation focused on simulation to model and analyse workflows on the cloud. As such, two research efforts were conducted. These are discussed below.

The initial research focused on the simulation-based analysis of internal IaaS behavioural knowledge alongside a workflow management system. Cloud workflow simulators do not provide sufficient support for the underlying virtualised infrastructure, such as physical machine state scheduling, virtual machine creation details and virtual machine placement. Other simulators are often user-centric and treat the cloud as a black box. Unfortunately, this behaviour prevents assessing the impact on the infrastructure of the various decisions made by the WMS. This dissertation presents DISSECT-CF-WMS, a workflow management system simulation built on DISSECT-CF. We developed DISSECT-CF-WMS to focus on the user-side behaviour of the clouds, while DISSECT-CF focuses on the internal behaviour of the IaaS systems. It enables better energy awareness by allowing the investigation of physical machine schedulers and customisable consumption characteristics. It also provides dynamic provisioning to meet the resource needs of the workflow application as it runs on the infrastructure, taking into account the provisioning delay of a VM in the cloud. It also provides a serverless simulation for executing scientific workflows based on the behaviour of real-world experiments of Amazon Lambda on DEWE v3.

We evaluated our simulator by running several workflow applications with different schedulers of physical machines for a given infrastructure. The experimental results show that workflow researchers can investigate different PM schedulers on infrastructure configurations to achieve lower energy consumption. The experiments also show that DISSECT-CF-WMS is up to 295$\times$ faster than WorkflowSim and still delivers accurate results. The experimental results of the auto-scaling mechanism show that the integration has the potential to optimise makespan, energy consumption, and VM utilisation over static provisioning. This work also allowed us to investigate Internal IaaS behavioural knowledge, such as different scheduling strategies for physical machines in a simulated environment; DISSECT-CF-WMS proved very useful. The

DOI: 10.14750/ME.2024.035

experimental results of our real-life experiments have validated the serverless simulation of DISSECT-CF-WMS.

We also presented a structure-aware scheduling algorithm for scientific workflows in the cloud with deadline constraints. When scheduling workflows in the cloud, resource allocation is important. A good resource estimation method helps the user to reduce the cost and time of workflow execution. Numerous algorithms face the challenge of meeting the user's deadline requirements while minimising the cost of running the workflow. The DSAWS scheduler presented in this dissertation analyses the structure of the incoming workflow and assigns an optimal resource provisioning mechanism based on the deadline constraint and the rank values of the tasks in the workflow. The main implementation of this algorithm is to make the second phase follow the schedule of the first phase (scheduling of workflow tasks on selected resources). We evaluate the performance of our algorithm by simulating it with four synthetic workflows based on real scientific workflows with different structures. For some structures (e.g., CyberShake and Epigenomics), our proposed algorithm uses the initial leased VMs to schedule all tasks of the same workflow to minimise data transfer costs. Other structures (e.g., Montage and LIGO) have many tasks with a short execution time, and many instances of the computation service are launched while only a small part of their time interval is used. Therefore, the proposed algorithm uses the remaining time in the current billing period of the VMs to avoid wasting resources. The proposed algorithm reduces the overall execution cost of a workflow while achieving a deadline set by the user. Experimental results show that DSAWS outperforms the Dyna and CGA algorithms in terms of meeting workflow deadlines while reducing execution costs. DSAWS met all the deadline factors of each workflow, while CGA and Dyna met 25% and 50%, respectively, of all the deadline factors of all workflows.

Although we have successfully achieved our original goals with our current research, some tasks still need to be done. Let us now discuss our future research direction.

5.1 Future Research Directions

In future works, we will extend the DISSECT-CF-WMS scheduling algorithm for dynamic provisioning to consider cost, makespan, resource utilisation, and energy consumption simultaneously. Multi-objective optimisation is a hot research area in workflow scheduling. Users might want to create algorithms for multi-objective scheduling optimisation by leveraging their knowledge of IaaS internals. These insights can optimise key workflow objectives (such as energy consumption, time, and resource utilisation). DISSECT-CF-WMS offers opportunities for scientific workflow applications that can be used for the upcoming research areas:

DOI: 10.14750/ME.2024.035

Resource Usage. Different allocation strategies of PMs to VMs can be developed to study their impact on performance, resource utilisation, energy consumption and fairness of workflows. More mechanisms could be added to reflect the environment in real life. An open research topic that cannot be discussed with commercial cloud companies because their VM placement technique is not public is the impact of background load on virtual resource performance. Performance degradation is always possible when multiple instances exist on a physical machine. If the network, memory or CPU become bottlenecks, virtual machine performance can be affected by this behaviour.

Data Centre Configurations. The DISSECT-CF simulator allows the properties of data centres (DCs) to be specified using the CloudLoader class. For our workflow executions, we can define both homogeneous and heterogeneous computing resources. Therefore, the base simulator allows the evaluation of different types of data centres to determine the best option for a specific type of workflow application. This allows the impact of the DC configuration on scientific workflow applications to be investigated through a series of experiments. The rest of the DISSECT-CF-WMS configuration remains the same, but the DC features change from one experiment to the next.

Moreover, we plan to improve our algorithm to consider not only the user's deadline but also other QoS objectives, such as resource utilisation and energy consumption, simultaneously. We also plan to extend our algorithm to support other cloud computing providers such as AWS Amazon and Azure. Finally, we plan to extend our algorithm to adapt to unforeseen delays resulting from the uncertainties of cloud frameworks.

DOI: 10.14750/ME.2024.035

5.2 Contributions to Science

This work contributes to the field of Cloud Computing, Distributed Systems and Scientific Workflows.

Thesis I: I proposed DISSECT-CF-WMS, a user-focused workflow simulation tool built upon an existing Infrastructure-as-a-Service simulation platform (DISSECT-CF). DISSECT-CF-WMS can run scientific workflow simulations and enable investigating internal IaaS behaviour. It also enables better energy awareness by investigating physical machine schedulers. It provides dynamic virtual machine provisioning to meet resource needs to execute scientific workflows, allowing WMSs to consider the provisioning delay of a VM in the cloud. DISSECT-CF-WMS also provides a serverless simulation for executing scientific workflows on AWS Lambda. Experimental results show that DISSECT-CF-WMS is up to 295 times faster than the competitor WorkflowSim simulation without affecting accuracy results. [**P1**, **P2**, **P3**, **P4**]

Thesis II: I proposed a new scheduling algorithm that optimizes resource provisioning based on the workflow structure, task ranking, and deadline constraints. The implementation ensures that the practical workflow execution follows the theoretical schedule, including the provisioning overheads. I evaluated the algorithm on four real-world scientific workflows with different structures. The algorithm's strength is to use the remaining time in the current virtual machine billing period at maximum to avoid wasting resources for other workflow structures and task execution time. Experimental results show that the proposed method outperforms the related algorithms regarding deadline satisfaction while reducing execution costs. [**P5**]

5.2.1 Author's Publications Related to the Dissertation

(P1) Al-Haboobi, Ali ; Kecskemeti, Gabor: "Reducing Execution Time of an Existing Lambda based Scientific Workflow System" In: The 12th Conference of PhD Students in Computer Science: Volume of short papers Szeged, Hungary : SZTE (2020) pp. 3-6. , 4 p. Scientific

(P2) Al-Haboobi, Ali ; Kecskemeti, Gabor: "Improving Existing WMS for Reduced Makespan of Workflows with Lambda" In: Euro-Par 2020: Parallel Processing Workshops : Euro-Par 2020 International Workshops, Warsaw, Poland: Springer (2021) 373 p. pp. 261-272. , 12 p. Web of Science (WoS), (Q2 Scopus Index), Impact Factor (0.969), (Conference paper) Scientific

(P3) Al-Haboobi, Ali ; Kecskemeti, Gabor: "Execution Time Reduction in Function Oriented Scientific Workflows" ACTA CYBERNETICA 25 : 2 pp. 131-150. , 20 p. (2021). Web of Science (WoS), (Q3 Scopus Index), Impact Factor (0.636), Journal Article

DOI: 10.14750/ME.2024.035

(P4) Al-Haboobi, Ali ; Kecskemeti, Gabor: ”Developing a Workflow Management System Simulation for Capturing Internal IaaS Behavioral Knowledge” JOURNAL OF GRID COMPUTING 21 : 1 Paper: 2 , 26 p. (2023). Web of Science (WoS), (Q1 Scopus Index), Impact Factor (4.111), Journal Article

(P5) Al-Haboobi, Ali ; Kecskemeti, Gabor: ”Structure-Aware Scheduling Algorithm for Deadline-Constrained Scientific Workflows in the Cloud” International Journal of Advanced Computer Science and Applications (IJACSA). Web of Science (WoS), (Q3 Scopus Index), Impact Factor (1.16), Journal Article. Accepted for publication.

5.2.2 Other Publications

(P6) András, Márkus ; Al-Haboobi, Ali ; Kecskeméti, Gábor ; Attila, Kertész: ”Simulating IoT Workflows in DISSECT-CF-Fog” SENSORS 23 : 3 Paper: 1294 , 16 p. (2023). Web of Science (WoS), (Q1 Scopus Index), Impact Factor (4.352), Journal Article

DOI: 10.14750/ME.2024.035

Bibliography

[1] Alex Abramovici, William E Althouse, Ronald WP Drever, Yekta Gürsel, Seiji Kawamura, Frederick J Raab, David Shoemaker, Lisa Sievers, Robert E Spero, Kip S Thorne, et al. Ligo: The laser interferometer gravitational-wave observatory. *science*, 256(5055):325–333, 1992.

[2] Saeid Abrishami, Mahmoud Naghibzadeh, and Dick HJ Epema. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. *Future generation computer systems*, 29(1):158–169, 2013.

[3] Sanjay P Ahuja and Bhagavathi Kaza. Performance evaluation of data intensive computing in the cloud. *International Journal of Cloud Applications and Computing (IJCAC)*, 4(2):34–47, 2014.

[4] Ali Al-Haboobi and Gabor Kecskemeti. Developing a workflow management system simulation for capturing internal iaas behavioural knowledge. *Journal of Grid Computing*, 21(1):2, 2023.

[5] Michael Albrecht, Patrick Donnelly, Peter Bui, and Douglas Thain. Makeflow: A portable abstraction for data intensive computing on clusters, clouds, and grids. In *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*, pages 1–13, 2012.

[6] Ilkay Altintas, Chad Berkley, Efrat Jaeger, Matthew Jones, Bertram Ludascher, and Steve Mock. Kepler: an extensible system for design and execution of scientific workflows. In *Proceedings. 16th International Conference on Scientific and Statistical Database Management, 2004.*, pages 423–424. IEEE, 2004.

[7] William H Bell, David G Cameron, A Paul Millar, Luigi Capozza, Kurt Stockinger, and Floriano Zini. Optorsim: A grid simulator for studying dynamic data replication strategies. *The International Journal of High Performance Computing Applications*, 17(4):403–416, 2003.

[8] Shishir Bharathi, Ann Chervenak, Ewa Deelman, Gaurang Mehta, Mei-Hui Su, and Karan Vahi. Characterization of scientific workflows. In *2008 third workshop on workflows in support of large-scale science*, pages 1–10. IEEE, 2008.

DOI: 10.14750/ME.2024.035

[9] James Blythe, Sonal Jain, Ewa Deelman, Yolanda Gil, Karan Vahi, Anirban Mandal, and Ken Kennedy. Task scheduling strategies for workflow-based applications in grids. In *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005.*, volume 2, pages 759–767. IEEE, 2005.

[10] Tracy D Braun, Howard Jay Siegel, Noah Beck, Ladislau L Bölöni, Muthucumaru Maheswaran, Albert I Reuther, James P Robertson, Mitchell D Theys, Bin Yao, Debra Hensgen, et al. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed computing*, 61(6):810–837, 2001.

[11] Duncan A Brown, Patrick R Brady, Alexander Dietz, Junwei Cao, Ben Johnson, and John McNabb. A case study on the use of workflow technologies for scientific analysis: Gravitational wave data analysis. In *Workflows for e-Science*, pages 39–59. Springer, 2007.

[12] Zhicheng Cai, Qianmu Li, and Xiaoping Li. Elasticsim: A toolkit for simulating workflows with cloud resource runtime auto-scaling and stochastic task execution times. *Journal of Grid Computing*, 15(2):257–272, 2017.

[13] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1):23–50, 2011.

[14] Junwei Cao, Stephen A Jarvis, Subhash Saini, and Graham R Nudd. Gridflow: Workflow management for grid computing. In *CCGrid 2003. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2003. Proceedings.*, pages 198–205. IEEE, 2003.

[15] Henri Casanova, Rafael Ferreira da Silva, Ryan Tanaka, Suraj Pandey, Gautam Jethwani, William Koch, Spencer Albrecht, James Oeth, and Frédéric Suter. Developing accurate and scalable simulators of production workflow management systems with wrench. *Future Generation Computer Systems*, 112:162–175, 2020.

[16] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, 74(10):2899–2917, 2014.

[17] Henri Casanova, Suraj Pandey, James Oeth, Ryan Tanaka, Frédéric Suter, and Rafael Ferreira da Silva. Wrench: A framework for simulating workflow man-

DOI: 10.14750/ME.2024.035

agement systems. In *2018 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*, pages 74–85. IEEE, 2018.

[18] Wei-Neng Chen and Jun Zhang. An ant colony optimization approach to a grid workflow scheduling problem with various qos requirements. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 39(1):29–43, 2008.

[19] Weiwei Chen and Ewa Deelman. Workflowsim: A toolkit for simulating scientific workflows in distributed environments. In *2012 IEEE 8th international conference on E-science*, pages 1–8. IEEE, 2012.

[20] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda. Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing*, 1:25–39, 2003.

[21] Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Sonal Patil, Mei-Hui Su, Karan Vahi, and Miron Livny. Pegasus: Mapping scientific workflows onto the grid. In *European Across Grids Conference*, pages 11–20. Springer, 2004.

[22] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira Da Silva, Miron Livny, et al. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, 46:17–35, 2015.

[23] Ann DiCaterino, Kai Larsen, Mei-Huei Tang, and Wen-Li Wang. An introduction to workflow management systems. Technical report, STATE UNIV OF NEW YORK AT ALBANY, 1997.

[24] Mahdi Ebrahimi, Aravind Mohan, and Shiyong Lu. Scheduling big data workflows in the cloud under deadline constraints. In *2018 IEEE Fourth International Conference on Big Data Computing Service and Applications (BigDataService)*, pages 33–40. IEEE, 2018.

[25] Hamid Reza Faragardi, Mohammad Reza Saleh Sedghpour, Saber Fazliahmadi, Thomas Fahringer, and Nayereh Rasouli. Grp-heft: A budget-constrained resource provisioning scheme for workflow scheduling in iaas clouds. *IEEE Transactions on Parallel and Distributed Systems*, 31(6):1239–1254, 2019.

[26] Kamil Figiela, Adam Gajek, Adam Zima, Beata Obrok, and Maciej Malawski. Performance evaluation of heterogeneous cloud functions. *Concurrency and Computation: Practice and Experience*, 30(23):e4792, 2018.

DOI: 10.14750/ME.2024.035

[27] Borivoje Furht, Armando Escalante, et al. *Handbook of cloud computing*, volume 3. Springer, 2010.

[28] Saurabh Kumar Garg and Rajkumar Buyya. Networkcloudsim: Modelling parallel applications in cloud simulations. In *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, pages 105–113. IEEE, 2011.

[29] Arash Ghorbannia Delavar and Yalda Aryan. Hsga: a hybrid heuristic algorithm for workflow scheduling in cloud systems. *Cluster computing*, 17(1):129–137, 2014.

[30] Robert Graves, Thomas H Jordan, Scott Callaghan, Ewa Deelman, Edward Field, Gideon Juve, Carl Kesselman, Philip Maechling, Gaurang Mehta, Kevin Milner, et al. Cybershake: A physics-based seismic hazard model for southern california. *Pure and Applied Geophysics*, 168(3-4):367–381, 2011.

[31] Adan Hirales-Carbajal, Andrei Tchernykh, Thomas Röblitz, and Ramin Yahyapour. A grid simulation framework to study advance scheduling strategies for complex workflow applications. In *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, pages 1–8. IEEE, 2010.

[32] Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine. Loggopsim: simulating large-scale applications in the loggops model. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 597–604, 2010.

[33] Joseph C Jacob, Daniel S Katz, G Bruce Berriman, John Good, Anastasia C Laity, Ewa Deelman, Carl Kesselman, Gurmeet Singh, Mei-Hui Su, Thomas A Prince, et al. Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking. *arXiv preprint arXiv:1005.4454*, 2010.

[34] L. Jhajj and S. Singh. A survey of workflow scheduling algorithms and research issues. *International Journal of Computer Applications*, 74, 2013.

[35] Qingye Jiang, Young Choon Lee, and Albert Y Zomaya. Serverless execution of scientific workflows. In *International Conference on Service-Oriented Computing*, pages 706–721. Springer, 2017.

[36] Gideon Juve and Ewa Deelman. Resource provisioning options for large-scale scientific workflows. In *2008 IEEE Fourth International Conference on eScience*, pages 608–613. IEEE, 2008.

DOI: 10.14750/ME.2024.035

[37] K Kanagaraj and S Swamynathan. Structure aware resource estimation for effective scheduling and execution of data intensive workflows in cloud. *Future Generation Computer Systems*, 79:878–891, 2018.

[38] Ali Husseinzadeh Kashan. League championship algorithm: a new algorithm for numerical function optimization. In *2009 international conference of soft computing and pattern recognition*, pages 43–48. IEEE, 2009.

[39] Gabor Kecskemeti. Dissect-cf: a simulator to foster energy-aware scheduling in infrastructure clouds. *Simulation Modelling Practice and Theory*, 58:188–218, 2015.

[40] Gabor Kecskemeti, Simon Ostermann, and Radu Prodan. Fostering energy-awareness in simulations behind scientific workflow management systems. In *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pages 29–38. IEEE, 2014.

[41] Joanna Kijak, Piotr Martyna, Maciej Pawlik, Bartosz Balis, and Maciej Malawski. Challenges for scheduling scientific workflows on cloud functions. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 460–467. IEEE, 2018.

[42] Madhu Sudan Kumar, Anubhav Choudhary, Indrajeet Gupta, and Prasanta K Jana. An efficient resource provisioning algorithm for workflow execution in cloud platform. *Cluster Computing*, pages 1–23, 2022.

[43] Hyungro Lee, Kumar Satyam, and Geoffrey Fox. Evaluation of production serverless computing environments. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 442–450. IEEE, 2018.

[44] Li Liu, Miao Zhang, Rajkumar Buyya, and Qi Fan. Deadline-constrained coevolutionary genetic algorithm for scientific workflow scheduling in cloud computing. *Concurrency and Computation: Practice and Experience*, 29(5):e3942, 2017.

[45] Jonathan Livny, Hidayat Teonadi, Miron Livny, and Matthew K Waldor. High-throughput, kingdom-wide prediction and annotation of bacterial non-coding rnas. *PloS one*, 3(9):e3197, 2008.

[46] Maciej Malawski, Adam Gajek, Adam Zima, Bartosz Balis, and Kamil Figiela. Serverless execution of scientific workflows: Experiments with hyperflow, aws lambda and google cloud functions. *Future Generation Computer Systems*, 2017.

DOI: 10.14750/ME.2024.035

[47] Maciej Malawski, Gideon Juve, Ewa Deelman, and Jarek Nabrzyski. Algorithms for cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. *Future Generation Computer Systems*, 48:1–18, 2015.

[48] Ming Mao and Marty Humphrey. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2011.

[49] Ming Mao and Marty Humphrey. A performance study on the vm startup time in the cloud. In *2012 IEEE Fifth International Conference on Cloud Computing*, pages 423–430. IEEE, 2012.

[50] Dejan Milojičić, Ignacio M Llorente, and Ruben S Montero. Opennebula: A cloud management tool. *IEEE Internet Computing*, 15(2):11–14, 2011.

[51] Aravind Mohan, Mahdi Ebrahimi, Shiyong Lu, and Alexander Kotov. Scheduling big data workflows in the cloud under budget constraints. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 2775–2784. IEEE, 2016.

[52] Alberto Nunez, Jose Luis Vazquez-Poletti, Agustin C Caminero, Jesus Carretero, and Ignacio Martin Llorente. Design of a new cloud computing simulation platform. In *International Conference on Computational Science and Its Applications*, pages 582–593. Springer, 2011.

[53] Simon Ostermann, Gabor Kecskemeti, and Radu Prodan. Multi-layered simulations at the heart of workflow enactment on clouds. *Concurrency and Computation: Practice and Experience*, 28(11):3180–3201, 2016.

[54] Simon Ostermann, Kassian Plankensteiner, Daniel Bodner, Georg Kraler, and Radu Prodan. Integration of an event-based simulation framework into a scientific workflow execution environment for grids and clouds. In *European Conference on a Service-Based Internet*, pages 1–13. Springer, 2011.

[55] Simon Ostermann, Kassian Plankensteiner, and Radu Prodan. Using a new event-based simulation framework for investigating resource provisioning in clouds. *Scientific Programming*, 19(2-3):161–178, 2011.

[56] Simon Ostermann, Kassian Plankensteiner, Radu Prodan, and Thomas Fahringer. Groudsim: An event-based simulation framework for computational grids and clouds. In *European Conference on Parallel Processing*, pages 305–313. Springer, 2010.

DOI: 10.14750/ME.2024.035

[57] Simon Ostermann, Radu Prodan, and Thomas Fahringer. Dynamic cloud provisioning for scientific grid workflows. In *2010 11th IEEE/ACM International Conference on Grid Computing*, pages 97–104. IEEE, 2010.

[58] Suraj Pandey, Linlin Wu, Siddeswara Mayura Guru, and Rajkumar Buyya. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In *2010 24th IEEE international conference on advanced information networking and applications*, pages 400–407. IEEE, 2010.

[59] P Rajasekar and Yogesh Palanichamy. Adaptive resource provisioning and scheduling algorithm for scientific workflows on iaas cloud. *SN Computer Science*, 2(6):1–16, 2021.

[60] Hajo A Reijers. *Design and control of workflow processes: business process management for the service industry*, volume 2617. Springer, 2003.

[61] Maria A Rodriguez and Rajkumar Buyya. Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms. *Future Generation Computer Systems*, 79:739–750, 2018.

[62] Maria Alejandra Rodriguez and Rajkumar Buyya. A taxonomy and survey on scheduling algorithms for scientific workflows in iaas cloud computing environments. *Concurrency and Computation: Practice and Experience*, 29(8):e4041, 2017.

[63] Sebastian Stadil, Scalr. Stadill s. by the numbers: How google compute engine stacks up to amazon ec2. `https://old.gigaom.com/2013/03/15/by-the-numbers-how-google-compute-engine-stacks-up-to-amazon-ec2/`, 2013. Accessed 12 Jul 2022.

[64] Domenico Talia. Workflow systems for science: Concepts and tools. *International Scholarly Research Notices*, 2013, 2013.

[65] Ian J Taylor, Ewa Deelman, Dennis B Gannon, Matthew Shields, et al. *Workflows for e-Science: scientific workflows for grids*, volume 1. Springer, 2007.

[66] Mustafa M Tikir, Michael A Laurenzano, Laura Carrington, and Allan Snavely. Psins: An open source event tracer and execution simulator for mpi applications. In *European Conference on Parallel Processing*, pages 135–148. Springer, 2009.

[67] Haluk Topcuoglu, Salim Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed systems*, 13(3):260–274, 2002.

DOI: 10.14750/ME.2024.035

[68] Meng-Han Tsai, Kuan-Chou Lai, Hsi-Ya Chang, Kuan Fu Chen, and Kuo-Chan Huang. Pewss: A platform of extensible workflow simulation service for workflow scheduling research. *Software: Practice and Experience*, 48(4):796–819, 2018.

[69] Jeffrey D. Ullman. Np-complete scheduling problems. *Journal of Computer and System sciences*, 10(3):384–393, 1975.

[70] Amandeep Verma and Sakshi Kaushal. Deadline constraint heuristic-based genetic algorithm for workflow scheduling in cloud. *International Journal of Grid and Utility Computing*, 5(2):96–106, 2014.

[71] Xin-She Yang. A new metaheuristic bat-inspired algorithm. In *Nature inspired cooperative strategies for optimization (NICSO 2010)*, pages 65–74. Springer, 2010.

[72] Jia Yu and Rajkumar Buyya. Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. *Scientific Programming*, 14(3-4):217–230, 2006.

[73] Amelie Chi Zhou, Bingsheng He, and Cheng Liu. Monetary cost optimizations for hosting workflow-as-a-service in iaas clouds. *IEEE transactions on cloud computing*, 4(1):34–48, 2015.

[74] Xiumin Zhou, Gongxuan Zhang, Jin Sun, Junlong Zhou, Tongquan Wei, and Shiyan Hu. Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based heft. *Future Generation Computer Systems*, 93:278–289, 2019.