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

Contents

1 Introduction

From the field of manufacturing and business processes, the workflow [24] 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 [27] are commonly used in several scientific fields, such as Montage [13] in astronomy, CyberShake [10] 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)[12, 28], Grid[5, 23] and Cloud[20]. 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[9], Kepler[4], and DEWE v3[14] 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 distributed 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 [14, 17, 18] have already demonstrated using FaaS for processing scientific workflows. DEWE v3 [14] 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 [30]. 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 [6, 20, 21] are often user-oriented and treat the cloud as a black box. Other workflow simulators [7, 11, 29] 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 WMSs. In contrast to the above problems, DISSECT-CF [16] 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 [14, 17, 18] 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 [25]. 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 [15]. Algorithms must decide when to provision or de-provision VMs depending on workflow requirements without violating the user's deadline.

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

2.1 The DISSECT-CF Workflow Management System

We implemented our WMS simulation approach on DISSECT-CF, a simulator focusing on internal infrastructure.

2.1.1 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 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. 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. DISSECT-CF provides three basic mechanisms for auto-scaling. 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 1. 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.

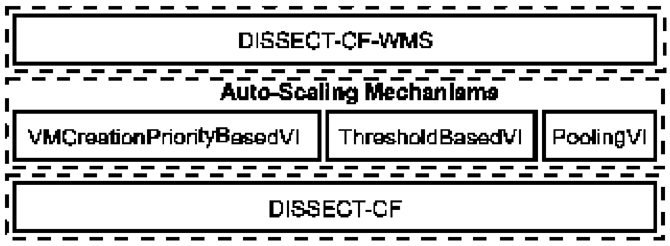


Figure 1: *The overview of the DISSECT-CF-WMS simulator integrated with the auto-scaling mechanisms of the DISSECT-CF simulator.*

2.1.2 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 behaviour in the following Section 2.1.3.

2.1.3 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 2 illustrates the steps of the original scheduling algorithm of DEWE v3.

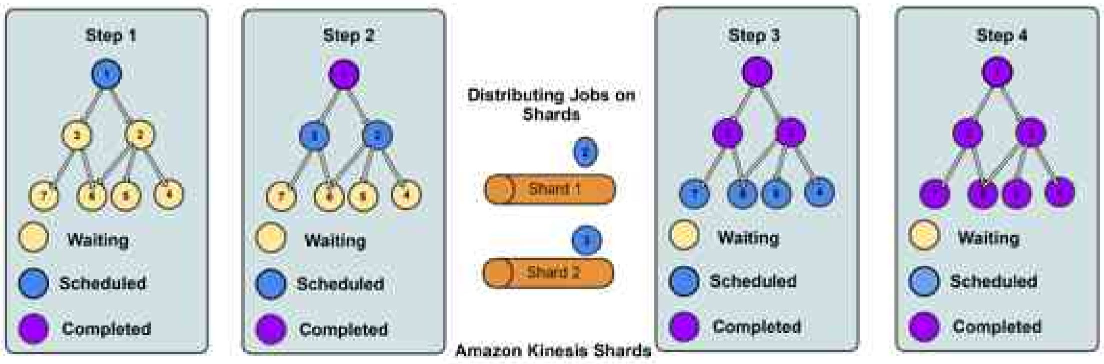


Figure 2: *The scheduling steps of the original algorithm with a sample workflow example.*

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 *SequenceNumber-ForOrdering*, which guarantees the order of jobs on a shard[1]. 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 illustrates the steps of the improved scheduling algorithm of DEWE v3.

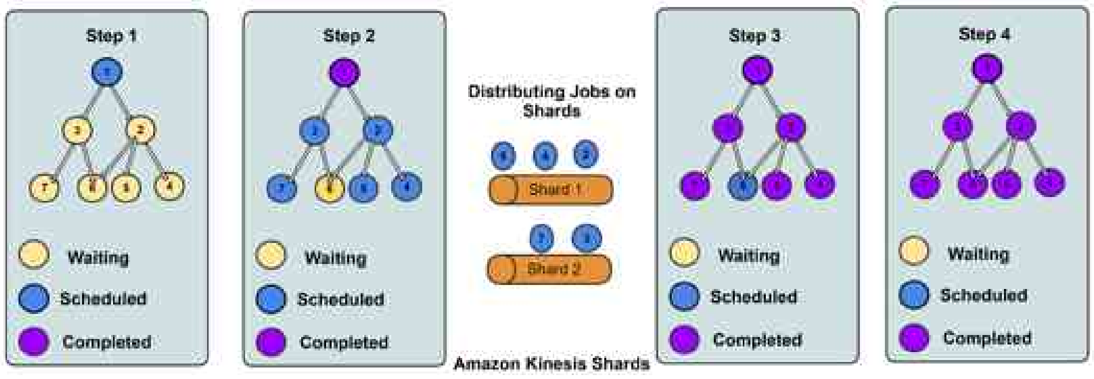


Figure 3: *The scheduling steps of the improved algorithm with a sample workflow example.*

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

2.1.4 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, 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). 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 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).

2.2 Evaluation

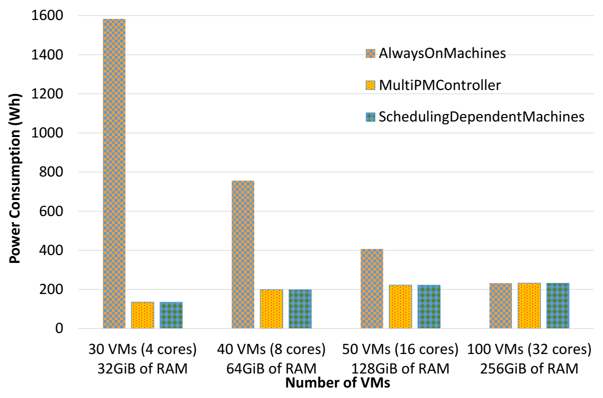
We demonstrate the capabilities of DISSECT-CF-WMS using the following evaluation experiments. First, we evaluated how the pre-existing three physical machine schedulers influence the energy consumption of various workflows. Second, we compared the simulation of DISSECT-CF-WMS with WorkflowSim regarding simulation accuracy and performance. Third, we have shown the advantages of using the auto-scaling mechanisms of DISSECT-CF-WMS to optimise makespan, energy consumption and VM utilisation over static provisioning. Finally, we evaluated the three built-in scheduling algorithms with four popular workflow applications in terms of energy consumption. 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.

2.2.1 Utilisation of Internal Cloud Infrastructure Details

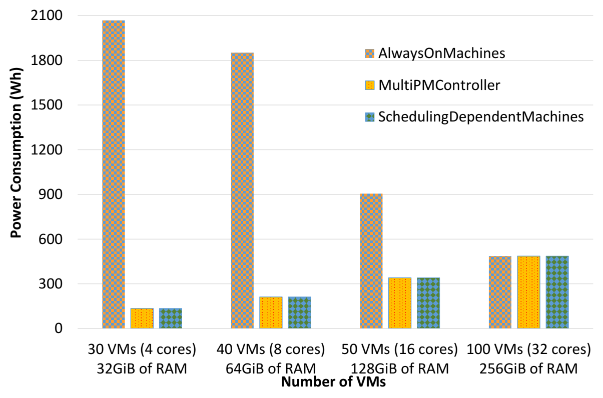
We configured a virtual infrastructure with a static number of VMs (in a single experiment, we set the number of VMs between 30 and 100; all VMs were homogeneous regarding the number of CPU cores and memory). We used FirstFitScheduler as a VM scheduler and the DataDependency algorithm as a task scheduler 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) SchedulingDependent-Machines (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 model for 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.

Figure 13 shows the collected energy consumption for each 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 achieve a large reduction in energy consumption, as SDM's strategy of switching on all machines in the data centre one by one (at a time) results in a long overall simulation time due to the provisioning delay, as shown in Figure 5. 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.

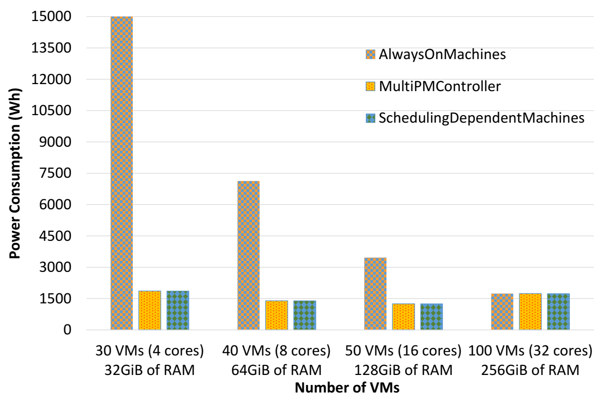
First, AOM has the same energy consumption patterns for all workflow applications because it never considers switching off machines, and thus it results in energy consumption for the entire infrastructure, even for the PMs that do not host VMs. Second, all PM schedulers have the same energy consumption when using the entire infrastructure, as they use all machines. Moreover, the MPMC and SDM schedulers have similar patterns



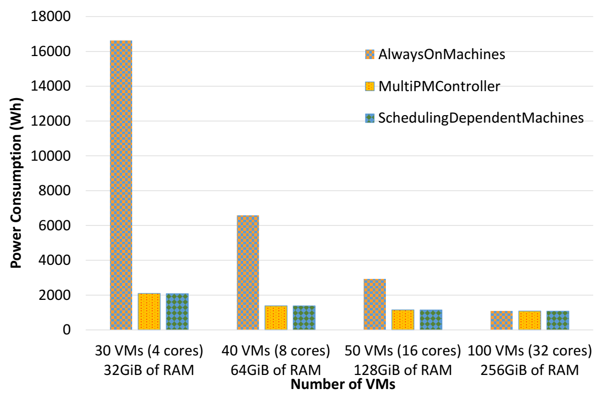
((a)) *Montage*



((b)) *CyberShake*



((c)) *Sipht*



((d)) *LIGO*

Figure 4: *The total power consumption of PM schedulers for four scientific applications on DISSECT-CF-WMS with different numbers of VMs.*

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, except for the Sipht experiment with 100 VMs, which only uses a maximum utilisation of VMs of 70% because Sipht has some tasks with significant differences in their running times so that the time difference can be as much as 19$\times$. In addition, In the second phase of Sipht, there are 64 jobs, while the number of VMs is 100. This results in idle time in other resources and scheduling gaps between tasks in the workflow, leading to the highest energy consumption. This pattern is repeated in LIGO's experiments, but the time difference can be as high as 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, as shown in Figure 5. 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 5. The differences are mainly because SDM is very slow in starting machines. 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 5). These differences show that switching on all PMs required for the workflow is advisable for dedicated private cloud infrastructures. 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. 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.

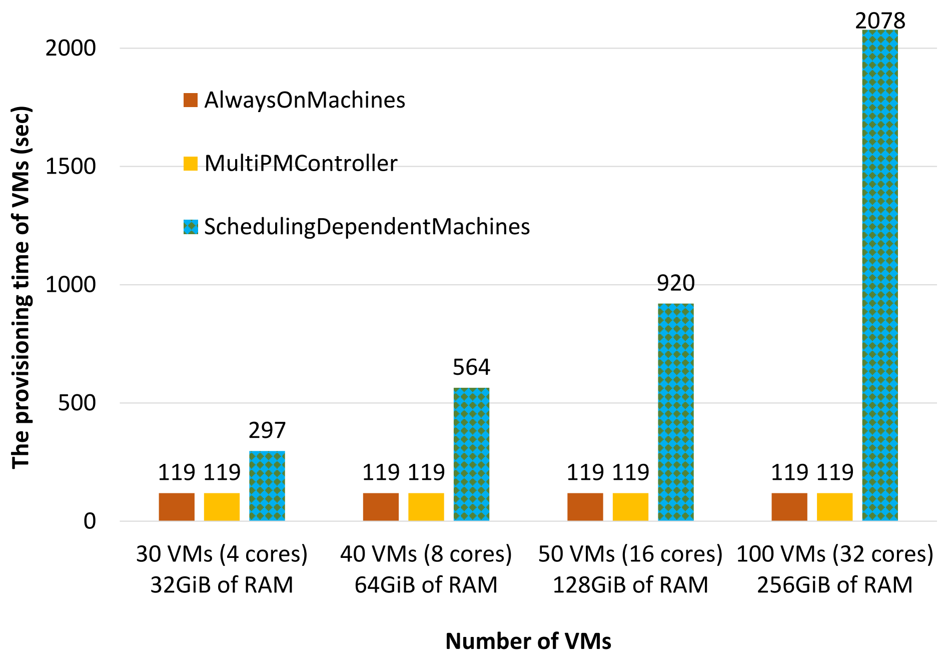


Figure 5: *The provisioning time of VMs for three PM schedulers on DISSECT-CF-WMS with different numbers of VMs.*

2.2.2 Simulation Times

Now that we have demonstrated the benefits that a WMS simulation extension can provide for the evaluation of 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 system with version 1.1.0 of WorkflowSim, using the same laptop as mentioned above. We present the results of the simulation in this section. 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 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 accurate and more 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 6 illustrates the performance differences between the simulators.

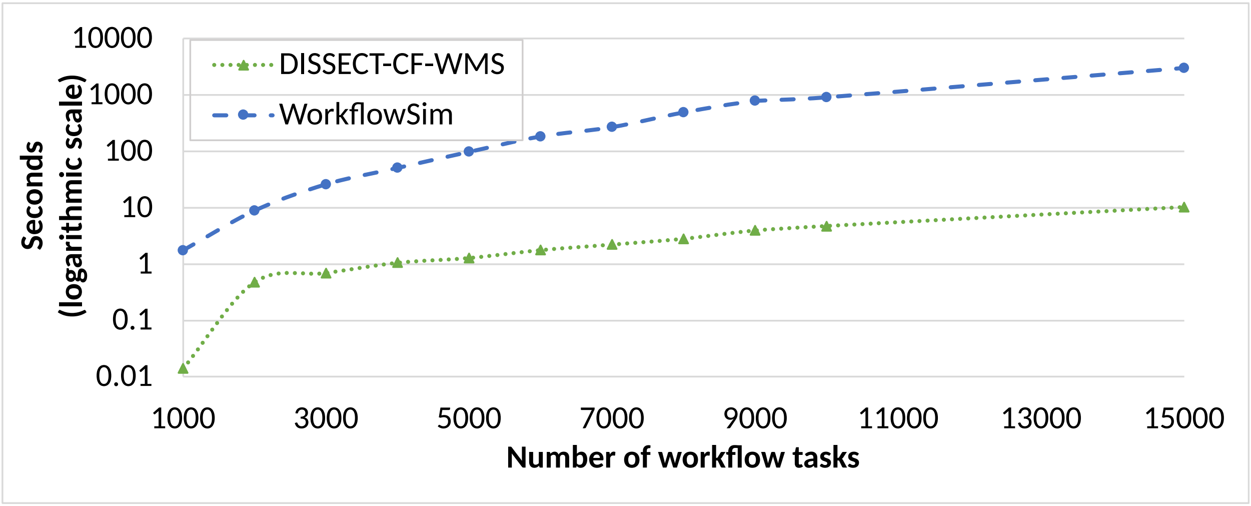


Figure 6: *The simulation time of DISSECT-CF-WMS and WorkflowSim simulators with different numbers of tasks in Montage workflows.*

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$\times$ (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[8], 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 [22]. DISSECT-CF, however, requires only one simulation thread (instead of one thread per entity). As a result, our WMS outperforms WorkflowSim, as shown in Figure 6.

2.2.3 Simulation versus Execution

The previous simulation experiment demonstrated the performance and accuracy of our system's simulation results to WorkflowSim. 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 [8]. 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 7 depicts Gantt charts of the real execution, whereas Figure 8 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 (executables) 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.

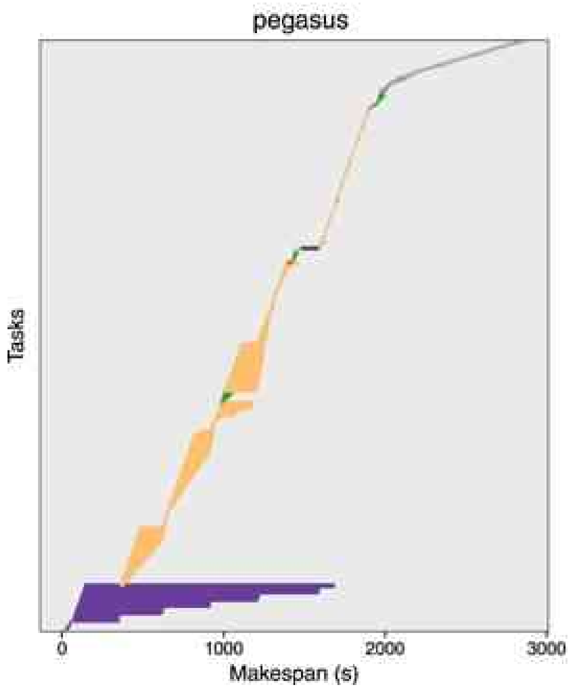


Figure 7: *Task execution Gantt chart for sample real-world ("pegasus") execution of the Montage-2.0 workflow on the AWS-m5.xlarge platform [8].*

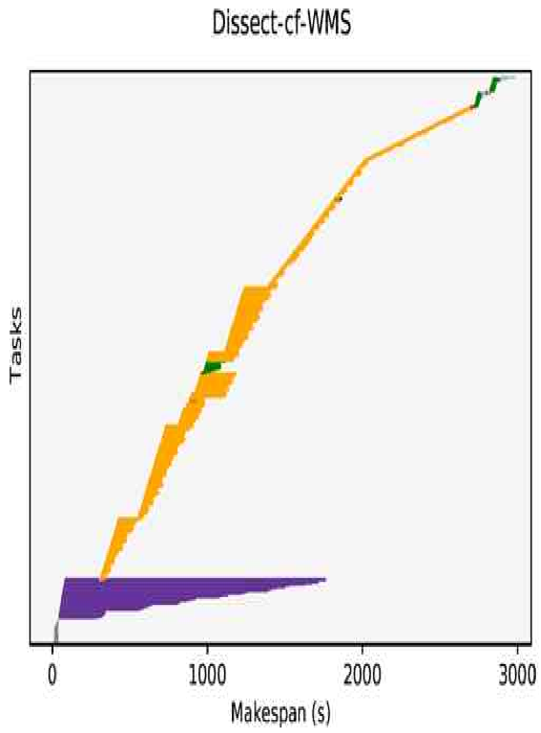


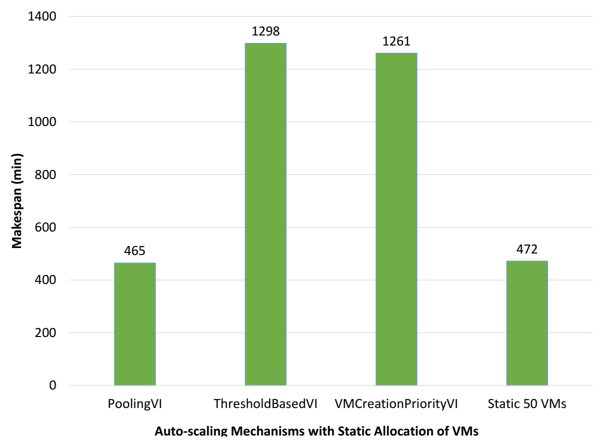
Figure 8: *Task execution Gantt chart for simulated Dissect-cf-WMS executions of the Montage-2.0 workflow on the AWS-m5.xlarge platform.*

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, you would need to develop and use the same data structures as the real implementation. This can be tedious or impossible depending on the data structures, languages, and/or libraries used. 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 scheduler, 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 7 and 8 can be attributed to this factor.

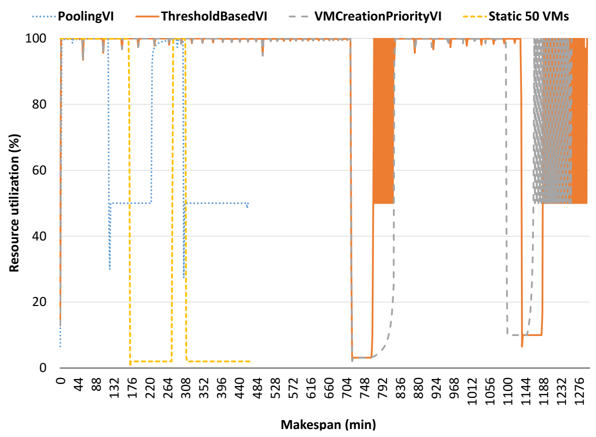
2.2.4 Auto-Scaling Mechanism

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. The virtual appliance repository can significantly reduce the time required to create virtual machines. In this experiment, we modified the Pooling VI to 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. 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.

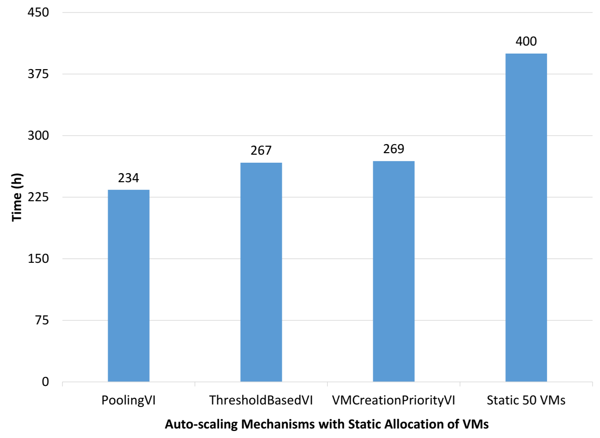
Figure 9(a) shows the results of executing the workflow. Pooling VI has the shortest total execution time compared to a dedicated cluster and the other auto-scaled virtual infrastructures. In terms of VM resource utilisation, Figure 9(b) also shows that pooling VI has the best average VM utilisation across all 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



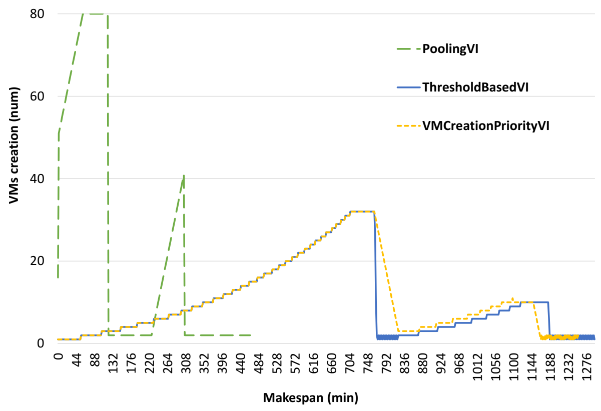
((a)) *Makespan*



((b)) *Resource consumption patterns*



((c)) *The total accounted for hours of VMs*

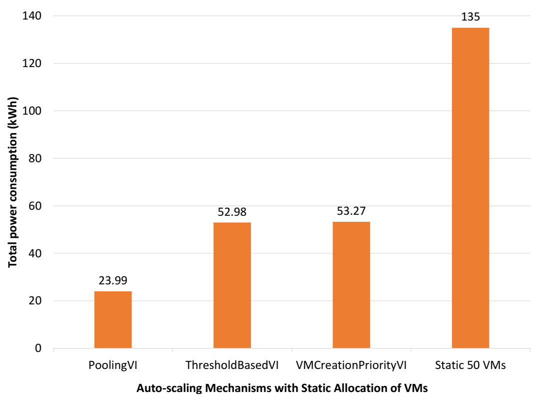


((d)) *The VMs creation patterns*

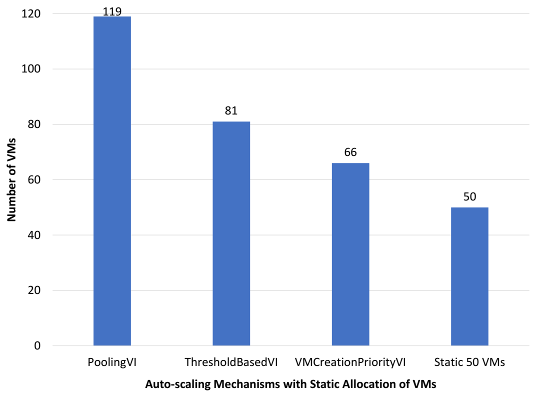
Figure 9: *The experiments of auto-scaling mechanisms and static 50 VMs with a large-scale Montage workflow (15000 tasks).*

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. However, the network bandwidth is fast enough to avoid bottlenecking the tasks that access the same central storage to store and retrieve the data files. Montage data sizes range from 4 to 1031 MB, with most being 4.2 MB, but a few are over 4.2 MB.

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 9(c). The Montage workflow consists of six single-threaded tasks executed sequentially 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 9(d)). 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



((a)) *Total power consumption (kWh)*



((b)) *Total accounted for the number of VMs (2 cores each)*

Figure 10: *The experiments of auto-scaling mechanisms and static 50 VMs with a large-scale Montage workflow (15000 tasks).*

number of VMs in response to a dynamic workload of workflows. In Figure 10(a), Pooling VI reduced energy consumption by more than 82% compared to static allocation. In addition, Pooling VI reduced energy consumption by about 54% compared to the other auto-scaling mechanisms. Although Pooling VI has used the most significant total number of VMs compared to static allocation and the other mechanisms (see Figure 10(b)), it has the lowest total number of hours billed, as shown in Figure 9(c).

2.2.5 Scheduling Experiments

In this experiment, we evaluate the energy consumption of three scheduling algorithms on DISSECT-CF-WMS without considering data transfers between jobs. We focused on energy consumption, which is one of the most important new objectives to optimise in the workflow scheduling area. Users can create algorithms for multi-objective scheduling optimisation by considering energy consumption alongside traditional cost and time objectives. Multi-objective optimisation is a hot research area in workflow scheduling. We tested the algorithms on the same cloud that we defined earlier. We tested the three algorithms with the workflow applications on ten VMs. The VMs were heterogeneous regarding the number of CPU cores, ranging from 1 to 10, with the first VM having one core and the last VM having 10 cores. We collected the energy consumption of DISSECT-CF-WMS as shown in Figure 11. We specifically note the need to collect energy consumption data for scheduling algorithms to evaluate their impact on energy consumption. We used the MPMC scheduler of the infrastructure and FirstFitScheduler as the VM scheduler. The algorithm HEFT has the best energy consumption over the other algorithms for all workflow applications except CyberShake, where MaxMin reduced energy consumption by 8% over HEFT. HEFT reduced the energy consumption of LIGO by 45% and 50% over MaxMin and Minmin, respectively. Also, HEFT has reduced the energy consumption of Sipht by 9% and 10% compared to MaxMin and MinMin, respectively.

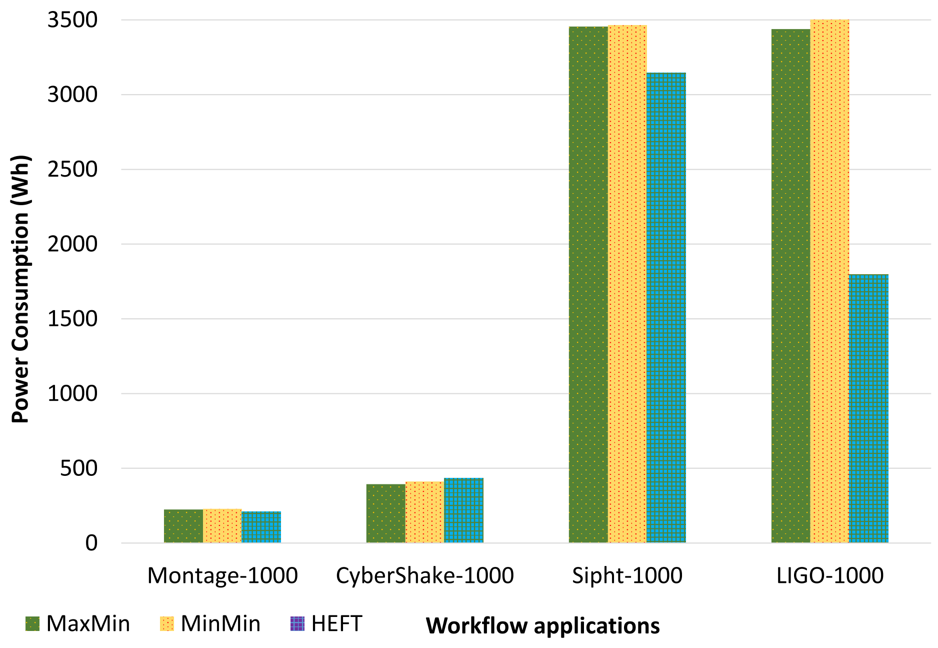


Figure 11: *The total power consumption of four workflow applications with three scheduling algorithms of DISSECT-CF-WMS on heterogeneous VMs.*

2.2.6 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 2.1.3 and 2.1.4.

2.2.7 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 is 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.

2.2.8 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:

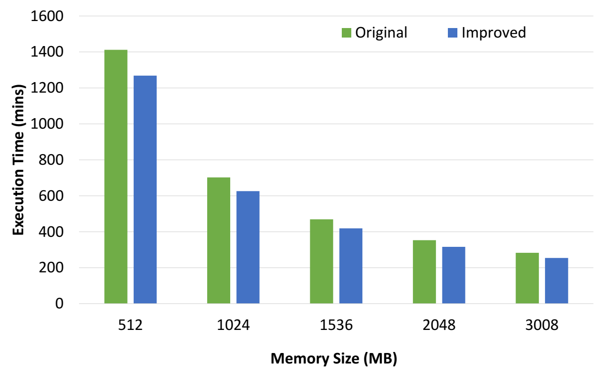
1. The Lambda Memory sizes were 512, 1024, 1536, 2048 and 3008 MB
2. The Lambda execution duration limit was limited to 900 seconds.
3. The batch size of the Lambda function was 20.
4. The number of Kinesis shards was set to 30.
5. One VM consisted of 4 CPU cores and 8 GiB of memory.

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. Figure 12 shows the total execution time (makespan) of Montage, CyberShake, Inspiral (LIGO) and Sipht workflows, respectively.

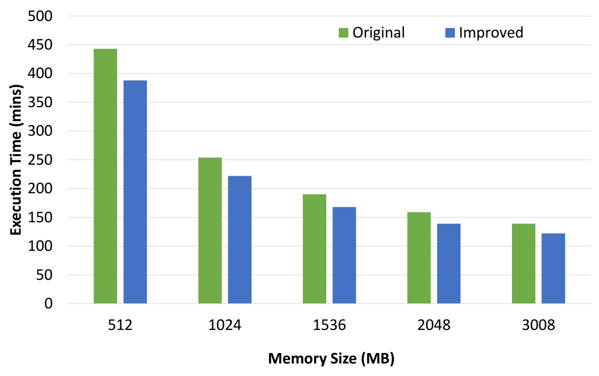
In the case of Montage, the improved algorithm schedules jobs with priority constraints to be executed in a single function invocation. Therefore, successor jobs can use the output files generated by their predecessor job in the same invocation. This has the potential to reduce workflow execution time by more than 10% compared to the original DEWE v3 algorithm, as shown in Figure 13(b). 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.

In the case of CyberShake, the improved algorithm reduced the makespan by 15% compared to the original DEWE v3 algorithm, as shown in Figure 14(b). Our improved 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.

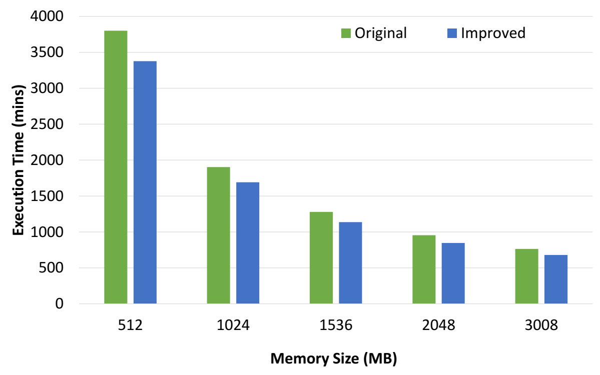
In the case of Inspiral (LIGO), the improved algorithm reduced the makespan by 11% compared to the original DEWE v3 algorithm, as shown in Figure 15(b). This pattern repeated (with almost gradual decreases) the same as for CyberShake. In LIGO, there are



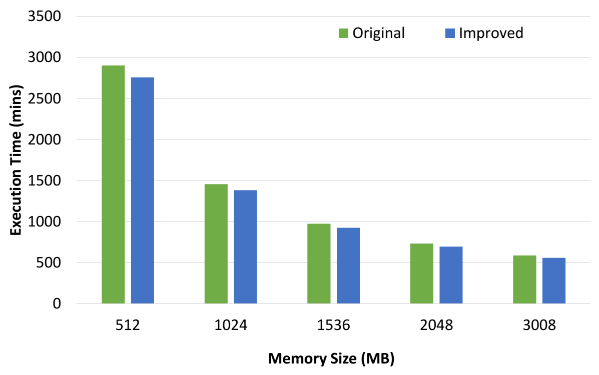
((a)) *Montage*



((b)) *CyberShake*



((c)) *LIGO*



((d)) *Sipht*

Figure 12: *The makespan of the two algorithms with four scientific applications running on different Lambda memory sizes.*

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 16(b). 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.

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 algorithm schedules jobs with precedence requirements into the same invocation. These results are consistent with what was expected for Montage's improved algorithm in Section 2.2.7.

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

3.1 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.

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.

The pseudocode of the entire DSAWS algorithm for workflow scheduling is shown in Algorithm 1. 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 1, it receives the workflow tasks arranged from Algorithm **??** 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 algorithm's execution. 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 1 selects ready tasks from the $rankList$ and adds them periodically to the $readyList$ in 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 algorithm's execution, no rented VMs are 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 2 in line 33.

Algorithm 2 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 2 represents the second phase, where workflow tasks are scheduled on the selected resources ($VMsList$) during the planning phase. It receives from Algorithm 1 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 and gaps in scheduling between workflow tasks. In lines 6 and 7, the algorithm 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.

Algorithm 1 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
```

Algorithm 2 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 $vm \in VMsList$ 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
```

3.2 Evaluation

Our experiment evaluated DSAWS with other competitive algorithms like CGA and Dyna for scheduling the selected scientific workflow applications. 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.

The experiment was conducted in the DISSECT-CF-WMS [3] 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.

Table 1: *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

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.

We created a model of the cloud infrastructure of Google Cloud Engine[2] 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. 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 [2]. 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 [26] and de-provisioning delay to 3 seconds [19] for all types of VMs. The bandwidth between VMs was set to 1 Gbit.

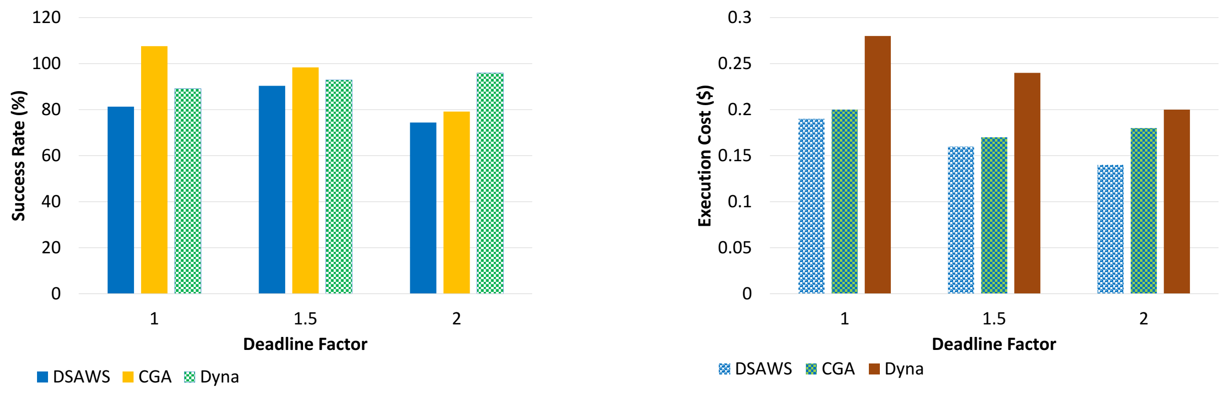
To evaluate the ability of each approach to achieve a valid solution that meets the deadlines, we set the success rate metric, which is calculated as the proportion of the current execution times to the given deadlines. For the evaluation, we set three deadline factors based on the maximum rank value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

Figures 13(a), 14(a), 15(a), and 16(a) show the results of the success ratios for each workflow with the three algorithms. On the other hand, Figures 13(b), 14(b), 15(b), and 16(b) show the execution costs (in $) for each workflow with the same algorithms.

In the case of Montage workflow, all algorithms completed the execution of the workflow within the deadline, except CGA, with the strict deadline factor, as shown in Figure 13(a). Dyna met all deadline factors, as shown in Figure 13(a). The DSAWS approach met all deadlines with the lowest cost compared to the other algorithms, as seen in Figure 13(b). 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 13(b). 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

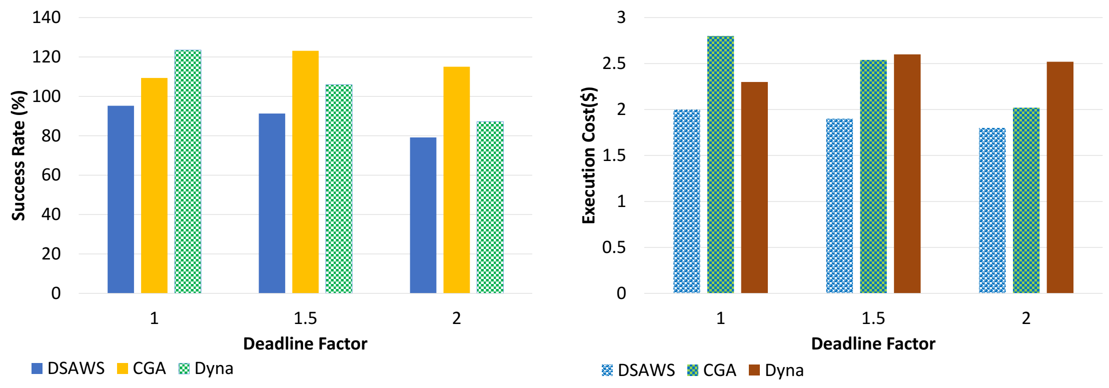
[2]`https://cloud.google.com/compute/all-pricing`



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

Figure 13: *The makespan and execution cost of the three algorithms with the Montage application.*

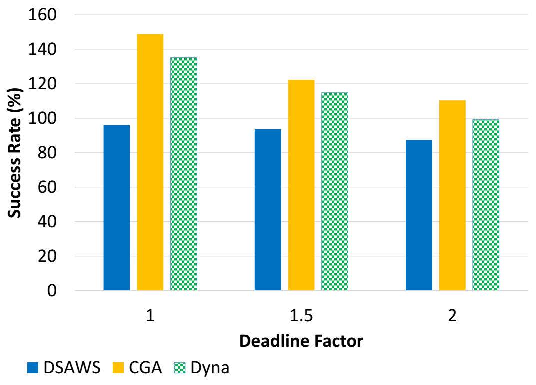
scheduling algorithms. This drawback is eliminated by the DSAWS described in this chapter, which allocates resources to all tasks based on their rank value. It guarantees that all tasks are completed within the deadline and starts new instances only when needed. Therefore, DSAWS reduces data transfer 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 with the lowest execution cost, as shown in Figure 14(a) and Figure 14(b), 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.



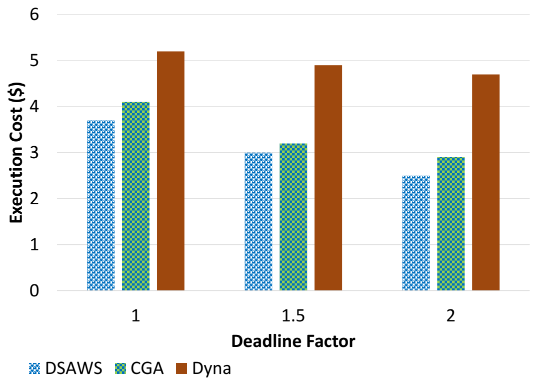
((a)) *Makespan* **((b))** *Cost*

Figure 14: *The makespan and execution cost of the three algorithms with the CyberShake application.*

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 15(a). 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 15(a). Also, unlike the other algorithms, DSAWS achieved the cheapest cost among all schedules, as shown in Figure 15(b). 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 and gaps in scheduling between workflow tasks in the case of CGA and Dyna.



((a)) *Makespan*

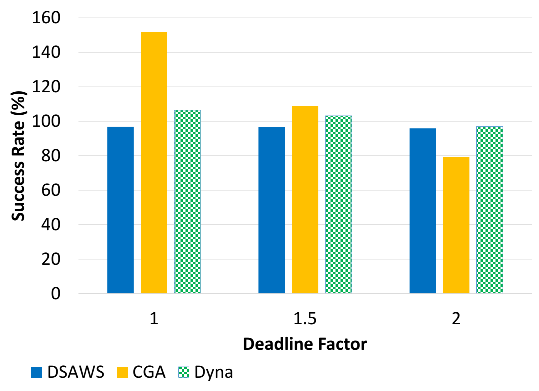


((b)) *Cost*

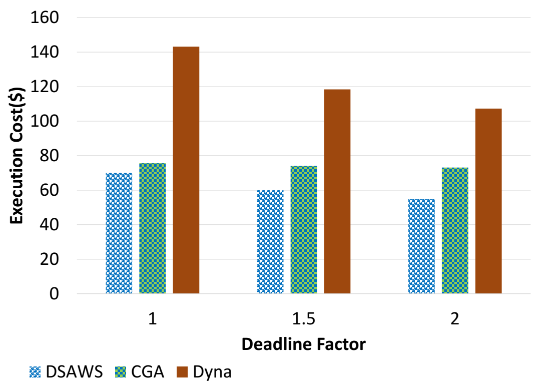
Figure 15: *The makespan and execution cost of the three algorithms with the LIGO application.*

In the case of 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 of 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 16(a). Furthermore, unlike the other two algorithms, DSAWS has the lowest execution cost, as shown in Figure 16(b). 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 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.

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



((a)) *Makespan*



((b)) *Cost*

Figure 16: *The makespan and execution cost of the three algorithms with the Epigenomics application.*

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 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.

4 Summary

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 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.

4.1 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**]

4.2 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

(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.

4.3 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

References

[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] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

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

[25] 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.

[26] 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.

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

[28] 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.

[29] 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.

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