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

Ali Al-Haboobi[1], Gabor Kecskemeti[2]
Institute of Information Technology, University of Miskolc, Miskolc, 3515, Hungary[1,2]
University of Kufa, Najaf, Iraq[1]

***Abstract*—Cloud computing provides pay-per-use IT services through the Internet. Although cloud computing resources can help scientific workflow applications, several algorithms face the problem of meeting the user's deadline while minimising the cost of workflow execution. In the cloud, selecting the appropriate type and the exact number of VMs is a major challenge for scheduling algorithms, as tasks in workflow applications are distributed very differently. Depending on workflow requirements, algorithms need to decide when to provision or de-provision VMs. Therefore, this paper presents an algorithm for effectively selecting and allocating resources. Based on the workflow structure, it decides the type and number of VMs to use and when to lease and release them. For some structures, our proposed algorithm uses the initial rented VMs to schedule all tasks of the same workflow to minimise data transfer costs. We evaluate the performance of our algorithm by simulating it with synthetic workflows derived from real scientific workflows with different structures. Our algorithm is compared with Dyna and CGA approaches in terms of meeting deadlines and execution costs. The experimental results show that the proposed algorithm met all the deadline factors of each workflow, while the CGA and Dyna algorithms met 25% and 50%, respectively, of all the deadline factors of all workflows. The results also show that the proposed algorithm provides more cost-efficient schedules than CGA and Dyna.**

***Keywords*—Workflow scheduling; workflow structure; cloud computing; resource provisioning; deadline constrained; infrastructure as a service**

I. Introduction

Cloud computing has become a significant platform for executing workflows as it allows the rental of resources on demand. It uses a pay-as-you-go billing model to provide IT resources over the internet [1]. This is done by renting virtual machines (VMs) with predefined CPU, memory, storage and network bandwidth capacities. To meet a wide range of application needs, customers can access various computing resources (i.e. VM sets) at different prices. Clouds offer infinite computing resources with different configurations that can be rented and used as needed. This architecture requires resource provisioning heuristics that run concurrently with a scheduling algorithm, which determines the amount and type of VMs to request from the cloud and the optimal time to rent and provision them.

Cloud computing today enables the execution of scientific applications consisting of hundreds or thousands of interdependent tasks [2]. Montage [3], CyberShake [4] and LIGO [5] are scientific workflow applications used in astronomy, earthquake science, and gravitational physics, respectively. A task does not begin its execution until all its predecessor tasks have been completed. Most of these scientific applications are built as workflows, which are groups of computational tasks linked by control and data dependencies. Each workflow phase consists of a different number of tasks, each requiring a different amount of computing resources. Depending on the application, a workflow can be extremely CPU-intensive and/or data-intensive. The complexity of task execution can vary from sequential execution to highly parallel execution with many inputs from different tasks.

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 [6]. In the cloud, faster and more powerful computing resources are often more expensive than slower ones. As a result, using powerful computing resources can increase execution costs by shortening workflow execution time. Consequently, the trade-off between time and cost is a major challenge for cloud-based workflow scheduling [7]. Two typical approaches are used to solve this: reducing the total execution time under a budget constraint [8] and reducing the financial cost under a time constraint [9]. This study 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.

Creating an optimal schedule in a heterogeneous cloud environment is NP-hard [10]. 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 [6]. 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 [11]. Moreover, it is necessary to determine when

these resources should be provisioned and when they should be de-provisioned during the workflow execution.

In this study, we present 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[12] and CGA[16].

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.

The remainder of the work is arranged as follows: Section II provides background information and reviews related work on workflow scheduling. The details of the design and implementation of the DSAWS algorithm are described in Section III. The experiment results are shown and discussed in section IV. Section V concludes the paper and future work.

II. Background Knowledge and Related Works

In this section, the presentation of scientific workflows is presented first. Then, related work on workflow scheduling with a problem statement on the clouds.

A. Background Knowledge

A workflow can be represented as a directed acyclic graph (DAG) consisting of a collection of atomic tasks. As shown in Fig. 1, the vertices of the workflow are a set of tasks $\{t_1, t_2, ..., t_n\}$, while the workflow edges represent data dependencies between these tasks. 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 data outputs become input dependencies for t_4. When t_4 is scheduled, its data input dependencies are sent to its target host to enable the successful execution of t_4.

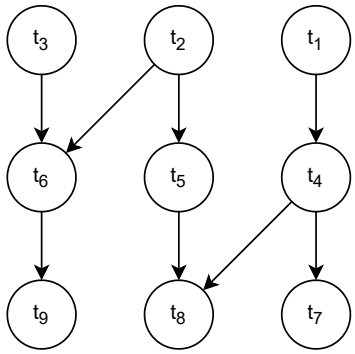

Fig. 1. A sample workflow.

B. 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) [13] and Particle Swarm Optimisation (PSO) [14] 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. [15] 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 [16] 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 [17], [18], [19], [20], [21] in the cloud computing environment are presented for workflow scheduling. Saeid et al. [20] 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. This is not practical during the period of process execution.

Xiumin et al. [22] have proposed a technique for extending HEFT [23]. 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) [16] 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 was chosen for comparison in our evaluation because of its static approach, which has the potential to generate optimal solutions. Nevertheless, our main interest is to compare DSAWS with CGA when both algorithms can meet deadlines. Moreover, by considering these algorithms, we can evaluate the adaptability of our results and show how DSAWS can meet deadlines while other algorithms fail to meet deadlines.

Dyna [12] 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 based on an A-star search to reduce costs. It is designed to schedule many workflows simultaneously but can also be modified to schedule only one. 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 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.

ARPS [24] 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, the experimental results show that they perform a workflow more effectively than other sophisticated algorithms to meet deadlines and reduce costs.

Mao et al. [25] 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, [30] 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 [26] 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 paper, we propose DSAWS, which is a complete full-ahead scheduling algorithm that considers the structure of the workflow. We discuss a method to deal with under- and over-provisioning issues.

III. The Proposed Scheduling Algorithm

Several objectives associated with task scheduling issues need to be addressed. The approach suggested in this paper 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.

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 I, 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

TABLE I. 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 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.
$readyList$	List of the ready tasks in workflow.
$rankList$	List of all tasks in descending order of their rank values.
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.

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
```

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.

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
```

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 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 3 The DSAWS scheduling algorithm

procedure DSAWS(G(T,E),D)
  m= available instance types of VMs (S)
  $rentedVMs = \varnothing$ the currently leased virtual machines
  $success$ = false.
  $vm_{booting}$ = the booting time of VM
  $vm_{minTime}$ = the earliest available time of vm in $rentedVMs$.
  $readyList$ = receives repeatedly ready tasks from $rankList$.
  $timeLine$ = represents the difference of subtracting $vm_{minTime}$ or t_{EST} from the deadline D.
  while (there exists unscheduled t in $readyList$) **do**
    t = find the earliest EST in $readyList$
    $vm_{minTime}$= find the earliest available time of vm in $rentedVMs$.
    $timeLine$:= D - $vm_{minTime}$
    for all $vm_j \in VM$ **do** where $j = 1, 2, \ldots, n$
      if $timeLine >= \frac{t_{rank}}{vm_j^{speed}}$ **then**
        select vm_j^{speed} to run t
        update EST for all successors of t
        remove t from $readyList$
        $success$:= true
      end if
    end for
    if success==false **then**
      $timeLine$:= D - t_{EST}
      for all $s_i \in S$ **do** where $i = 1, 2, \ldots, m$
        if $timeLine >= (\frac{t_{rank}}{s_i^{speed}})$ **then**
          select a new instance vm_i^{speed} to run t
          remove t from $readyList$
          update EST for all successors of t
          add vm_i^{speed} to $rentedVMs$
        end if
      end for
    end if
  end while
  call TimelineVMs(VMs)
end procedure

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.

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** $vm \in VMsList$ **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**

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.

A. An Illustrative Example

To illustrate how the proposed algorithm works, we apply its steps to a sample workflow shown in Fig. 2. 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.

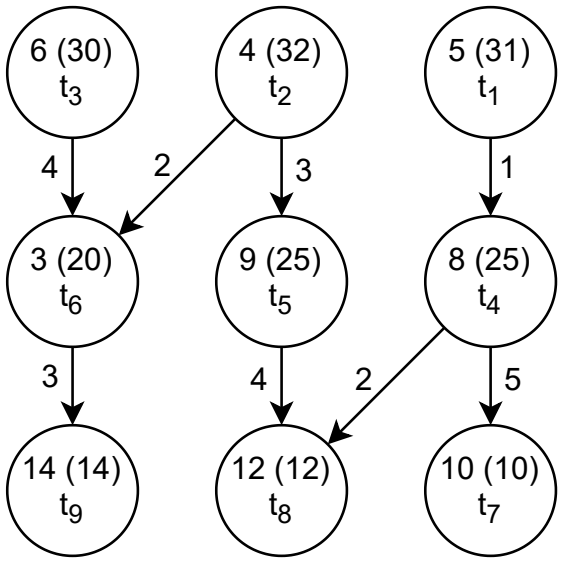

Fig. 2. A sample workflow.

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 Fig. 2, we call the DSAWS scheduling algorithm, 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 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 II 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 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 called 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 II) 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.

TABLE II. THE SCHEDULING OF THE WORKFLOW TASKS FOR EACH STEP OF EXECUTING DSAWS ON THE SAMPLE WORKFLOW OF FIG. 2

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

TABLE III. THE CHARACTERISTICS VALUES FOR EACH WORKFLOW APPLICATION

Workflow type	Number of levels	Number of tasks	Number of dependencies	Mean runtime (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

IV. EVALUATION

Our experiment evaluated DSAWS with other competitive algorithms like CGA and Dyna for scheduling the selected scientific workflow applications. The experiment was conducted in the DISSECT-CF-WMS [27] 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.

We used a library of realistic workflows introduced by Bharathi et al. [28] to evaluate our scheduling algorithm. We evaluate our algorithm by simulating it with synthetic workflows based on real scientific workflows with different structures. We selected four realistic workflows from different scientific applications, which are Montage from the field of astronomy, CyberShake from the field of earthquake science, Inspiral (LIGO) from the field of gravitational physics, and Epigenomics from the field of biology. Fig. 3 shows the structure of each workflow. All relevant characteristic values required for the above algorithms are listed in Table III for the analysis of experiments. We have used these values to obtain the rank values and assign the corresponding VM to each task. The performance of the four workflows in DSWAS is compared with Dyna and CGA approaches.

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 IV 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

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

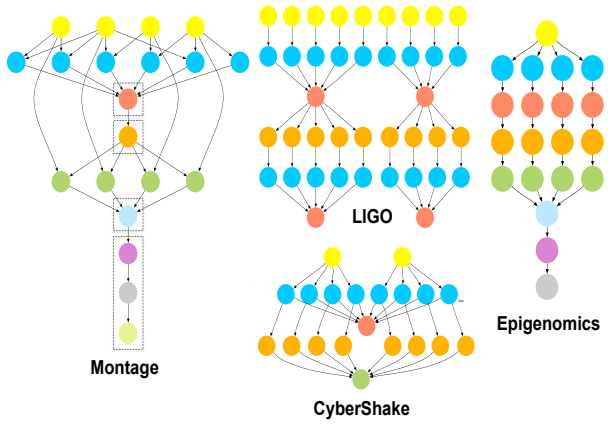

Fig. 3. The structure of the Montage, CyberShake, LIGO and Epigenomics workflows [28].

TABLE IV. 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

[29]. 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 [31] and de-provisioning delay to three seconds [25] 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 V. In addition, the moderate and relaxed deadlines are obtained by multiplying the maximum rank values by (1.5) and (2), respectively.

Fig. 4, 6, 8, and 10 show the results of the success ratios for each workflow with the three algorithms. On the other hand,

TABLE V. 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

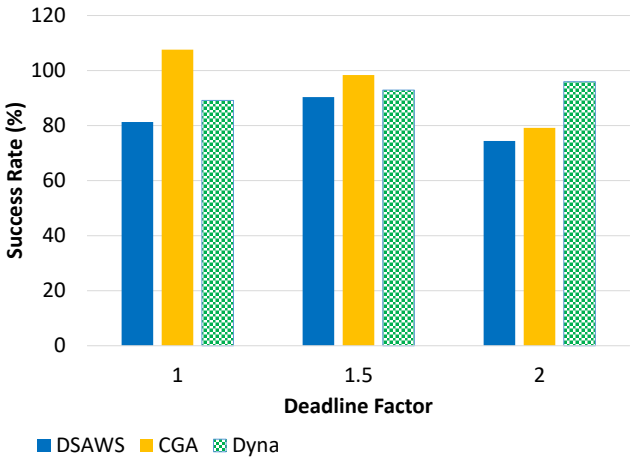

Fig. 4. The makespan of the three algorithms with the montage application.

Fig. 5, 7, 9, and 11 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 Fig. 4. Dyna met all deadline factors, as shown in Fig. 4. The DSAWS approach met all deadlines with the lowest cost compared to the other algorithms, as seen in Fig. 5. 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 Fig. 5. 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 paper, 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 Fig. 6 and Fig. 7,

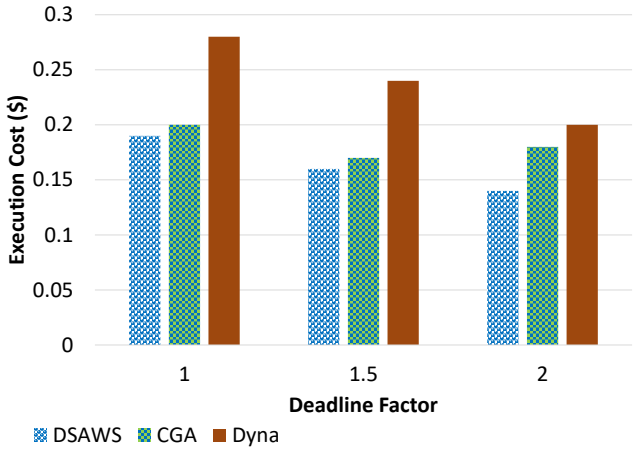

Fig. 5. The execution cost of the three algorithms with the montage application.

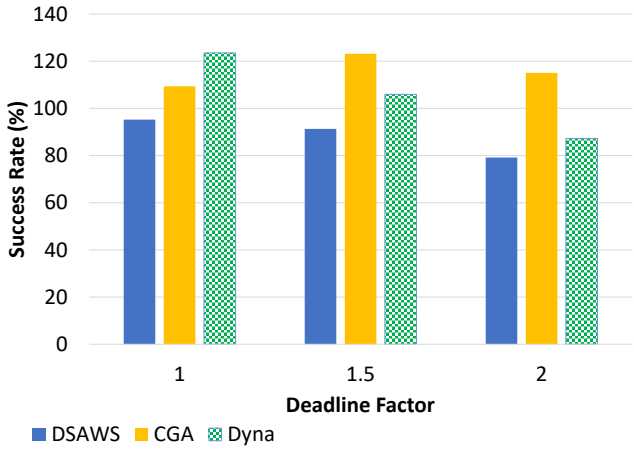

Fig. 6. The makespan of the three algorithms with the CyberShake application.

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.

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 Fig. 8. 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

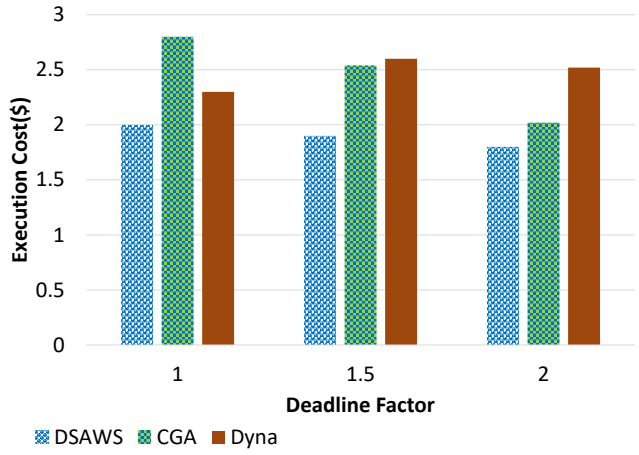

Fig. 7. The execution cost of the three algorithms with the CyberShake application.

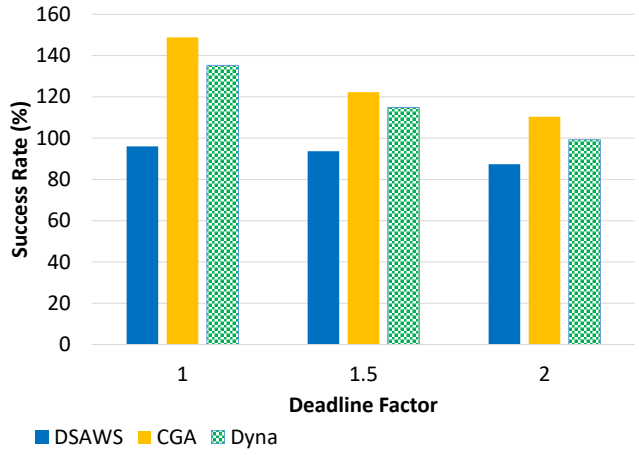

Fig. 8. The makespan of the three algorithms with the LIGO application.

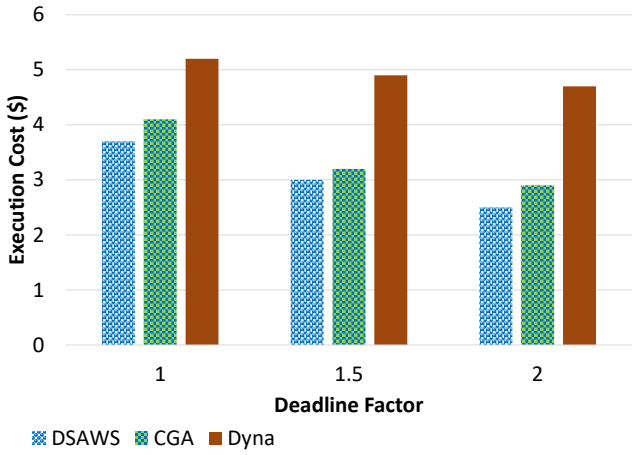

Fig. 9. The execution cost of the three algorithms with the LIGO application.

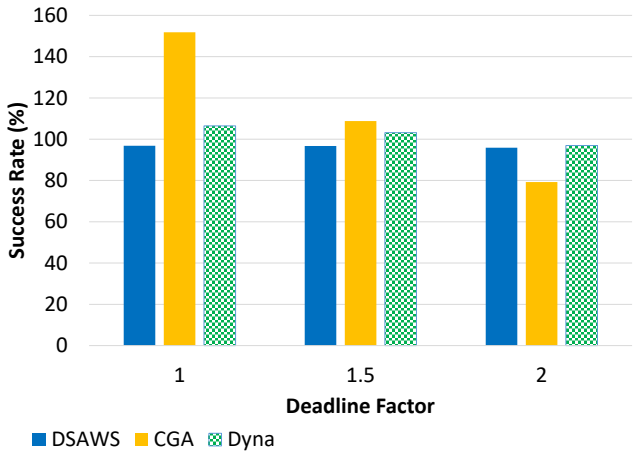

Fig. 10. The makespan of the three algorithms with the Epigenomics application.

in the workflow and executes them to meet the user-specified deadline, as shown in Fig. 8. Also, unlike the other algorithms, DSAWS achieved the cheapest cost among all schedules, as shown in Fig. 9. 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.

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 Fig. 10. Furthermore, unlike the other two algorithms, DSAWS has the lowest execution cost, as shown in Fig. 11. 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 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, regardless of whether the deadline was met or not. Moreover, CGA showcases its ability to generate more cost-effective schedules and surpasses Dyna about 92% regardless of whether the deadline was met or not. For some structures (e.g., CyberShake and Epigenomics),

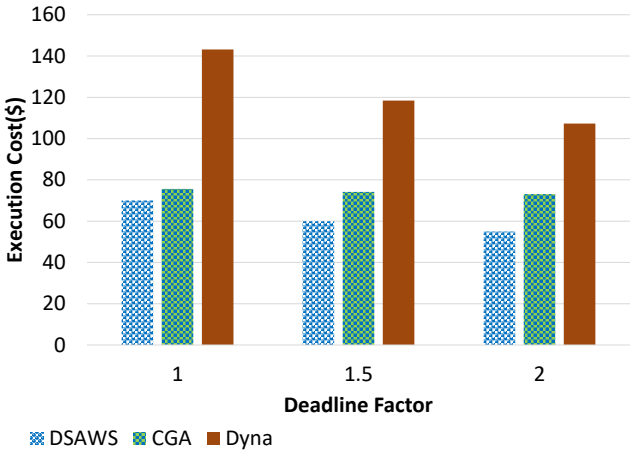

Fig. 11. The execution cost of the three algorithms with the Epigenomics application.

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.

V. Conclusion and Future Works

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 paper 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.

In the future, we plan to improve our algorithm to consider the user's deadline and other Quality of Service (QoS) objectives, such as resource utilisation and energy consumption, simultaneously.

Conflict of Interest: The authors declare that they have no conflict of interest.

References

[1] Jyoti Sahni and Deo Prakash Vidyarthi. A cost-effective deadline constrained dynamic scheduling algorithm for scientific workflows in a cloud environment. IEEE Transactions on Cloud Computing, 6(1):2–18, 2015.

[2] Wenzhong Guo, Bing Lin, Guolong Chen, Yuzhong Chen, and Feng Liang. Cost-driven scheduling for deadline-based workflow across multiple clouds. IEEE Transactions on Network and Service Management, 15(4):1571–1585, 2018.

[3] 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.

[4] 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.

[5] Alex Abramovici, William E Althouse, Ronald WP Drever, Yekta Gursel, 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.

[6] 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.

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

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

[20] 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.

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

[22] 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.

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

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] Sebastian Stadil, Scalr. Stadill s. by the numbers: How google compute engine stacks up to amazon ec2, 2013.