Konjaang and Xu *Journal of Cloud Computing: Advances, Systems and Applications* (2021) 10:11
https://doi.org/10.1186/s13677-020-00219-1

Journal of Cloud Computing:
Advances, Systems and Applications

Research Open Access

Multi-objective workflow optimization strategy (MOWOS) for cloud computing

J. Kok Konjaang* and Lina Xu

Abstract

Workflow scheduling involves mapping large tasks onto cloud resources to improve scheduling efficiency. This has attracted the interest of many researchers, who devoted their time and resources to improve the performance of scheduling in cloud computing. However, scientific workflows are big data applications, hence the executions are expensive and time consuming. In order to address this issue, we have extended our previous work "Cost Optimised Heuristic Algorithm (COHA)" and presented a novel workflow scheduling algorithm named Multi-Objective Workflow Optimization Strategy (MOWOS) to jointly reduce execution cost and execution makespan. MOWOS employs tasks splitting mechanism to split large tasks into sub-tasks to reduce their scheduling length. Moreover, two new algorithms called MaxVM selection and MinVM selection are presented in MOWOS for task allocations. The design purpose of MOWOS is to enable all tasks to successfully meet their deadlines at a reduced time and budget. We have carefully tested the performance of MOWOS with a list of workflow inputs. The simulation results have demonstrated that MOWOS can effectively perform VM allocation and deployment, and well handle incoming streaming tasks with a random arriving rate. The performance of the proposed algorithm increases significantly in large and extra-large workflow tasks than in small and medium workflow tasks when compared to the state-of-art work. It can greatly reduce cost by 8%, minimize makespan by 10% and improve resource utilization by 53%, while also allowing all tasks to meet their deadlines.

Keywords: Multi-objective, Cloud computing, Execution cost, Makespan, Scientific workflows

Introduction

Cloud computing, a multipurpose and high-performance internet-based computing, can model and transform a large range of application requirements into a set of workflow tasks. It allows users to represent their computational needs conveniently for data retrieval, reformatting, and analysis [1]. Over the past decades, researchers from different scientific domains such as astronomy, physics, earth science, and bioinformatics have used cloud platforms to model scientific applications for many real-world problems. These applications are modeled as workflows [2] which allow complex and large scientific data to be analyzed and simulated in a cloud computing environment. This is because cloud computing has lower the upfront capital expenditure on hardware, software, hosting, and deployment [3]. It presents enormous opportunities that allow workflow applications to be scheduled at a reduced cost and time [4].

The cloud provides infinite resources which are accessible via network on a pay-as-you-go basis [5, 6]. These infinite resources have made cloud computing a unique selling proposition hub in the IT sector. This has inspired tremendous researches leading to the deployment of highly technological platforms such as the internet of things and mobile edge computing. These platforms, through cloud computing, can create a smart environment that provides smart healthcare, cities, transportation, housing, energy, living, and many more, to facilitate our way of living. Cloud computing can model various applications as a set of workflow tasks [7]. These workflow tasks have a set of edges which represent the data

*Correspondence: james.konjaang@ucdconnect.ie
University College Dublin, School of Computer Science, Dublin, Ireland

dependencies between the workflows. In other words, it indicates that a successor workflow task cannot start until the predecessor workflow task is completed [2, 8–10].

Scheduling workflows in the cloud computing environment is gaining ground and remains an attractive research area for many scientists. This is attributed to the rapid growth of the cloud industry and the opportunities it present for cloud users. The cloud can deploy resources virtually or remotely, which allows scientific discoveries to be carried out on a large scale [11]. However, generating an effective schedule with the current heuristics algorithms remains a challenge. Scientific workflows are big data applications and often require a large budget and more time to execute. This is due to their nature and data size. This problem becomes more obvious when the workflow tasks to be scheduled have deadlines. Much work has been done by other researchers to find an optimal solution to this problem through heuristic algorithms. Nevertheless, the problem still exists. This is because most of these heuristics rely heavily on job priority without considering the scheduling length of the job. Hence, it is very difficult to achieve an optimal solution with the current heuristic algorithms.

Moreover, the workflow scheduling, in particular, is complicated and has been defined by many researchers as NP-complete problem, thus making the orchestration of workflow tasks execution challenging [12, 13]. This is due to the complexity in the structure of scientific workflows, as one workflow application can produce many discrete tasks for scheduling [14]. As a result, generating a schedule to optimize the two most important, yet conflicting scheduling objectives i.e., execution cost and execution makespan becomes a complicated problem. For example, optimizing execution cost increases the execution makespan. This is due to the interlink that exists between these objectives. Makespan and cost optimization problem persist because VM selection which is a key in managing resource utilization to improve system throughput is usually ignored by researchers. Execution cost and makespan conflicting challenge is an acknowledged problem that needs to be addressed appropriately [15, 16].

In this paper, we addressed the workflow scheduling problem by extending our work originally presented in IEEE 6th Internal Conference on Big Data Security on Cloud (BigDataSecurity) [17]. In our previous work, a task splitting algorithm known as Cost Optimised Heuristic Algorithm (COHA) [17] was used to split large tasks with longer executing lengths to allow them to meet their deadlines at a lesser cost. However, in the previous work, we only applied execution cost as the performance evaluation metric which is not adequate to measure the efficiency of the algorithm. Moreover, there exist some research gaps such as VM selecting and task mapping criteria that are worthy of further investigation. We extended our previous work to consider tasks execution makespan and resource utilization as metrics for performance evaluation and optimization goals.

The main contributions of the paper are summarized as follows:

1 We introduce a triple-stage layer workflow execution model and cloud resources model to support achieve the aim of the proposed algorithm.
2 We presented a multi-objective workflow minimization strategy (MOWOS) to jointly minimize the execution cost and execution makespan of workflow tasks.
3 A novel measure called MaxVM selection is introduced. This method is responsible for selecting and mapping workflow tasks with maximum (longer) execution time on VMs with Maximum (higher) execution capacities. This is done to help reduce the waiting times of workflow tasks with longer execution times.
4 An efficient scheme known as MinVM selection method is introduced to select and map workflows with minimum (shorter) execution time. This is done to avoid mapping smaller workflows on VMs with higher execution capacities that comes with higher cost and may lead to an increase in execution cost.
5 Re-evaluating the variants of the extended algorithm through four real scientific workflows.

The remaining paper is structured as follows. Related work is introduced in "Related work" section. "System models" section described our system models, a detailed description of the proposed Multi-Objective Workflow Optimization Strategy (MOWOS) is presented in "Proposed algorithm: multi-objective workflow optimization strategy (MOWOS)" section. The performance Evaluation, Experimental Setting, Workflow structure, Results and Analysis are presented in "Performance evaluation" section, and finally, the paper is concluded in "Conclusion and future work" section.

Related work

Workflow scheduling is one most difficult task which needs to be looked at in the cloud computing environment. This is due to the complexity in its structure, as one workflow application can produce many discrete tasks for scheduling [14]. This have been defined by many researchers as NP-complete problem [13, 18–24]. Considerable research efforts have been made by previous researchers to solve the workflow scheduling problem, nevertheless, the problem persists. For example, in [25], the NP-complete problem was proven. The researchers transform non-convex constraint to many linear constraints using linearization and reformulation based on heuristic techniques. On the other hand, the researchers

in [26] presented GA-ETI to consider the relationship between jobs and their required data to enhance the efficiency in running workflow tasks on cloud resources. GA-ETI is capable of optimizing both makespan and cost. However, it is restricted by prior knowledge from identifying overloaded and under-loaded VM for workload redistribution.

Different types of research, based on the Min-Min scheduling algorithm for task scheduling has been conducted to reduce makespan, execution cost, and to improve the utilization rate of cloud resources. For example, Liu et al. [27] took into consideration three task scheduling constraints such as quality of service, the dynamic priority model and the cost of service; and proposed an improved Min-min algorithm for task scheduling in a cloud computing environment, for enhanced makespan and resource utilization rate. The results show that the improved approach is efficient and can increase the utilization rate of cloud resources. Also, it can schedule large tasks timely to meet the requirements of cloud users. However, it is less effective when there are more large tasks than short tasks. Also, the researchers in literature [28–31] have acknowledged the impressive performance of min-min in reducing makespan and have compared their methods with min-min and other existing algorithms to ascertain the performance of their methods concerning execution makespan and execution cost.

Many researchers have also used Max-Min in different capacities to enhance task scheduling in the cloud computing environment. For example, Li et al. [32] presented an improved Max-Min based technique call MIN-Max-Min algorithms. It reduces the average makespan of jobs. However, the proposed method is not efficient to exploit parallel tasks from multiple sources, and hence not able to reduce idle time slots. Also, the method is not scalable and does not consider the dynamic nature of cloud resources. Ghumman and Kaur [33] presented a hybrid method called improved Max-Min Ant Colony Algorithm. The method combines the concept of max-min and ant colony algorithm to get workflow scheduled. Through simulations, the proposed method is seen to be efficient in providing better results in makespan and execution cost. However, the approach does not consider the length of workflows and VM selection methods, and therefore could not fully utilize the available resources effectively.

Also, in [8], a Fuzzy Dominance sort based Heterogeneous Earliest-Finish-Time (FDHEFT) algorithm was presented. The approach has two phases, thus, task prioritizing phase and instance selection phase. In the task prioritizing phase, the algorithm calculates the priorities of every task and queue them in non-increasing order with the upper values ranked first. The instance selection phase sorts and selects all tasks based on their fuzzy dominance (priority) values to minimize cost and makespan. However, the limitation of FDHEFT is that, tasks are selected based on their fuzzy dominance values without considering the size of task and its corresponding VM speed. Matching workflow task to the appropriate resource for execution will help avoid task missing their deadline. A general framework heuristic algorithm for multi-objective static scheduling of scientific workflows in heterogeneous computing environments call Multi-objective list scheduling algorithm (MOLS) was proposed in [34]. The algorithm tries to find a suitable Pareto solution by deploying two strategies, that is, maximizing the distance to the user constraint vector for dominant solutions and minimizing it otherwise. Though, proposed algorithm is capable of producing better results, in cost and makespan, the approach mainly focuses in reducing makespan and cost without considering the workload and it impact on resources.

Besides, Cost-Effective Deadline Constrained Dynamic scheduling algorithm for scientific workflow scheduling in cloud known as Just-In-Time (JIT-C) was proposed in [35]. JIT-C rely on the many advantages presented by cloud, while taking care of the performance differences in VMs and instance acquisition delay for effective scheduling to meet deadline of all workflow task at a reduced makespan and cost. The algorithm addresses three major issues including VM performance variation, resource acquisition delays and heterogeneous nature of cloud resources. Also, the issue of runtime overhead of the algorithm was not left out. Other methods such as: (i) Pre-processed approach for combining pipeline tasks in a single task to save data transfer time and reduces the runtime overhead, (ii) Monitor control loop technique to monitors the progress of all running workflow tasks and makes scheduling decision in terms of performance variation and (ii) Plan and Schedule method to coordinate with 'cheapesttaskVMmap' method for low cost schedule were deployed. Though the proposed method is proven to be effective and efficient in meeting deadlines, producing low makepsan and cost, however, it is very expensive in generating schedules when the deadline factor is low. At a reduced deadline factor, the slack time is likely to be zero or low and when this happen it can increase the cost of executing a workflow task.

Other different researches based on reducing execution cost and energy consumption under deadline constraints are considered. In this regard, Li et al 2015 suggested a cost-effective energy-aware scheduling algorithm for scientific workflows in heterogeneous cloud computing environments. The proposed method is intended to minimize the execution cost of workflow and reduce the energy consumption while meeting the deadlines of all workflow tasks. To achieve this, four different methods were deployed which include: i) the VM selection algorithms

that use the concept of cost-utility to map workflow tasks onto the best VMs. ii) Task merging methods to minimize execution cost and energy consumption, iii) VM reuse method to reuse the unused VM instance and iv) Task slacking algorithm based on DVFS techniques to save energy of leased VMs. Cost-effective energy-aware algorithms can minimize the execution cost of workflows and considerably save energy. However, the proposed method consumes more time to identify VMs types. This can affect the execution cost of workflow tasks since time is a major determiner of cost in the cloud computing environment.

Moreover, Haidri et al. [22] identified VM acquisition delay as one main challenge for workflow task scheduling in a cloud computing environment. They proposed a Cost-Effective Deadline Aware (CEDA) scheduling strategy to optimize total workflow task execution time and economic cost, while meeting deadlines. The method selects a workflow task with the highest upward rank value at each step and dispatches it to the cheapest instance for a reduced makespan and cost. Also, slack time was used to schedule other tasks to further reduce the price overhead. However, CEDA is not effective for for large workflows. In [36], Customer Facilitated Cost-based Scheduling (CFCSC) algorithm was presented. The method is presented to schedule a task to reduce cost and execution makespan on the available cloud resources. CFCSC is only efficient with small workflow task but performs abysmally in makespan when large numbers of tasks are scheduled. This can be attributed to the fact that CFCSC assigns workflow tasks in a critical path to cloud resources and allowing the non-critical path workflows to stay long in the queue.

From the views of all the researchers in the literature, it is observed that most of the methodologies focus on resource efficiency to optimize workflow scheduling. This can cause load imbalance and inefficient resource utilization [37]. Different from the aforementioned work, our study presents a task splitting management system that considers both resource efficiency and the workload to be scheduled. Considering the complexity of workflows, we provided Maxvm and Minvm allocation strategies to reduce the system execution cost, time and to fully utilize the cloud resources, while ensuring tasks meet their deadlines.

System models

Workflow application model

A scientific workflow is a representation of a set of workflow tasks which is modeled as a directed acyclic graph (DAG) [38], and defined by a tuple G = (W, E). Where W = (Wt_1, Wt_2, Wt_3, Wt_4 ……., Wt_n); Wt is a set of 'n' workflow tasks in a scientific workflow application. Where E donate the set of edges which represent the flow of data dependencies constraint between workflow task Wt_i and workflow task Wt_j which is denoted by $E_{i,j}$ = (Wt_i,Wt_j). Every edge $E_{i,j}$ is a representation of a precedence constraint workflow which indicates that workflow task wt_j cannot start until wt_i completes. In this scenario, workflow task wt_i is a predecessor workflow task of wt_j while workflow task wt_j is called the immediate successors of workflow task wt_i. On this note, all predecessor workflow task of wt_i is represented as $pre_{(wti)}$ while all the successor workflow task is represented as $succ_{(wti)}$. Therefore predecessor workflow task and successor's workflow tasks are donated by Eqs. 1 and 2:

$$Pre\,(Wt_i) = \left\{Wt_j | (Wt_j, Wt_i) \epsilon D\right\} \tag{1}$$

$$Succ\,(Wt_i) = \left\{Wt_j | (Wt_j, Wt_i) \epsilon D\right\} \tag{2}$$

Every workflow DAG has an entry task and exit task. Figure 1 is a representation of a sample workflow DAG of 12 tasks with entry and exit tasks. An entry workflow task is a workflow task without a predecessor workflow task which is donated by wt_{entry} as in Eq. 3.

$$Pre(Wt_{entry}) = \phi \tag{3}$$

An exit workflow task is a workflow task without a successor, which is also donated by wt_{exit} as in Eq. 4.

$$Succ(Wt_{exit}) = \phi \tag{4}$$

Resource model

Resource allocation involves the management of cloud resources in the cloud datacenter to increase system efficiency. The resource model consists of several cloud users and different cloud service providers as in Fig. 2. Let CSP = (csp_1, csp_2, csp_3 ..... csp_n) be the list of Cloud Service Providers offering cloud resources (VMs), and let r = $\bigcup_{s=1}^{\infty}\{r_s\}$ represent the available VM in the cloud data center which is unlimited for the cloud user. Let K = $\bigcup_{k=1}^{n}\{R_k\}$ denote the types of VM where n is the number of VM in type k [8] which are represented as R = (vm_1, vm_2, vm_3 .....vm_k) be the list of cloud resources available to a list of cloud users represented by cu = (cu_1, cu_2, cu_3 ..... cu_n) for workflow task execution. These VMs have different configurations and different prices and are modelled by a tuple vm(pc; c) [39], where *pc* represents the processing capacity of the VM and *c* denotes the monetary cost of the VM which is payable in hourly bases. Each resource in the resource list has a unique configuration and the billing is based on the processing capacity of the VM. In other-words, a VM with higher processing capacity cost more than a VM with lesser processing capacity [40]. Let *pc* = (pc_{min}, pc_{max},.....Pc_{xx}) be the processing

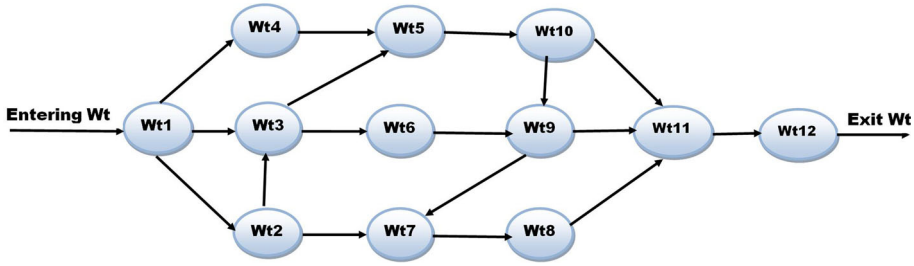

Fig. 1 Sample workflow dag with entering and exit workflow tasks

capacities of the various VMs in the VM list. For example, Amazon EC2 offers VM instances like Micro, Small, Medium, and Large [41] as deployed in this research. Each of these VMs are connected with a communication link as c^l and a bandwidth as b^w. In this model, we assume there are only two categories of VMs available for tasks mapping expressed in Eqs. 5 and 6.

$$Mapping\ 1 = \left\{ If\ ECT_{wt1}^{VMk} = Dl_{wt1}\ then\ wt_1 \ni VM_{max} \right\} \tag{5}$$

$$Mapping\ 2 = \left\{ If\ ECT_{wt1}^{VMk} < Dl_{wt1}\ then\ wt_1 \ni VM_{min} \right\} \tag{6}$$

Where ECT is the expected completing time of wt_1 on VM_k, Dl is the deadline of wt_1. The two categories of VMs used in this research are defined below:

Category 1

$$\Leftrightarrow \mathrm{VM}_{min} \begin{cases} Micro \Leftrightarrow (Processor\ 250MIPS) \Leftrightarrow (cost\ per\ hour\ 0.15) \\ Small \Leftrightarrow (Processor\ 500MIPS) \Leftrightarrow (cost\ per\ hour\ 0.3) \end{cases}$$

and

Category 2

$$\Leftrightarrow \mathrm{VM}_{max} \begin{cases} Medium \Leftrightarrow (Processor\ 1000MIPS) \Leftrightarrow (cost\ per\ hour\ 0.6) \\ Large \Leftrightarrow (Processor\ 2000MIPS) \Leftrightarrow (cost\ per\ hour\ 0.9) \end{cases}$$

Workflow execution model

Mostly, scientific workflows are used to manage data flow. It is modeled as a directed acyclic graph (DAG) which represents a sequence of tasks that processes a set of data [2, 8]. The execution process of workflow has two major phases which include, the resource provisioning phase and task generating and mapping phase [2]. The resource provisioning phase discovers all the available cloud computing instances (both hardware and software) and deploys them to guarantee a successful execution of every incoming task. The tasks mapping phase, on the other hand, is a process where all the unmapped tasks in the metadata are mapped onto the various Virtual Machines (VMs) for execution. The aim of executing a workflow is to ensure Efficient, Effective and Just-in-time Scheduling Plan (EEJSP) that will increase throughput, minimize the makespan, and total execution cost [42].

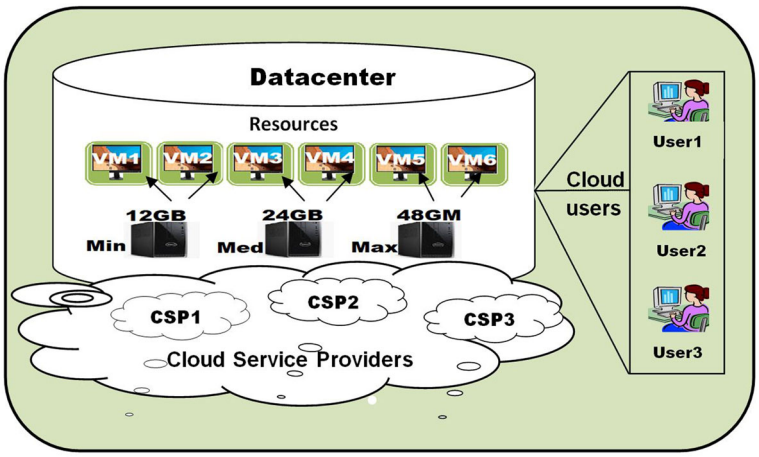

Fig. 2 Cloud resource model

Our proposed workflow execution model presented in Fig. 3 is a triple-stage layer model that relies on the opportunities and challenges of cloud computing while taking into account workflows deadlines and QoS constraints. The first layer is the application layer, followed by the execution Layer, and the cloud infrastructure layer. The layers of the workflow execution model used in this work are highlighted below:

1 **Application layer**: The application layer provides the types of workflow applications which is used as a data set for this research. For simulation purposes, four real-world workflows provided by the Pegasus workflow management system [43–45] are used. These workflows have been practically used by different researchers to model and evaluate the performance of workflow scheduling algorithms in the cloud computing domain. These applications are modeled as a DAG with edges $e(wt_i, wt_j)$ between the workflow tasks. The edges of the workflows represent precedence constrains. The edge $e(wt_i, wt_j)$ indicates that wt_i is a direct predecessor of wt_j and should finish execution before workflow wt_j which is an immediate successor of wt_i [5]. These workflows are applied in different scientific domains such as bioinformatics, astronomy, astrophysics, etc. [42, 45].
2 **Execution Layer:** In scheduling, a range of workflow tasks with different sizes, with or without scheduling constraints arrived at the execution layer for scheduling decisions to be made. The execution layer comprising the proposed scheduler (MOWOS); the constraints such as deadlines, budget, and quality of service; and the optimization objectives, is responsible for ensuring that every arrived workflow task is given the opportunity to be mapped onto a VM by the scheduler. The job of the scheduler is to ensure that these workflow tasks are successfully scheduled at a given budget and time. For example, a workflow task wt_i arrived at the execution layer with a deadline of 2 min. In this scenario, workflow task wt_i needs to be mapped onto a VM before its deadline as specified by the user. In this case, the scheduler will have to select a VM with the capacity to schedule wt_i such that, it will not violate the deadline constraint.

The primary aim of the execution layer is to manage all incoming workflow tasks, to find an optimal solution to two important, yet conflicting scheduling objectives such as execution makespan and execution cost. Makespan is defined as the maximum finishing time among all received workflows per time. It shows the quality of the workflows assignment on VMs from the execution time point of view. The cost of execution workflow task (wt_i on vm_k) is defined as $Cost(wt_i, vm_k) = ET(wt_i, vm_k) \times Cost(vm_k)$ [40], where $ET(wt_i, vm_k)$ is the Execution Time for executing workflow task wt_i on vm_k and cost (vm_k) is

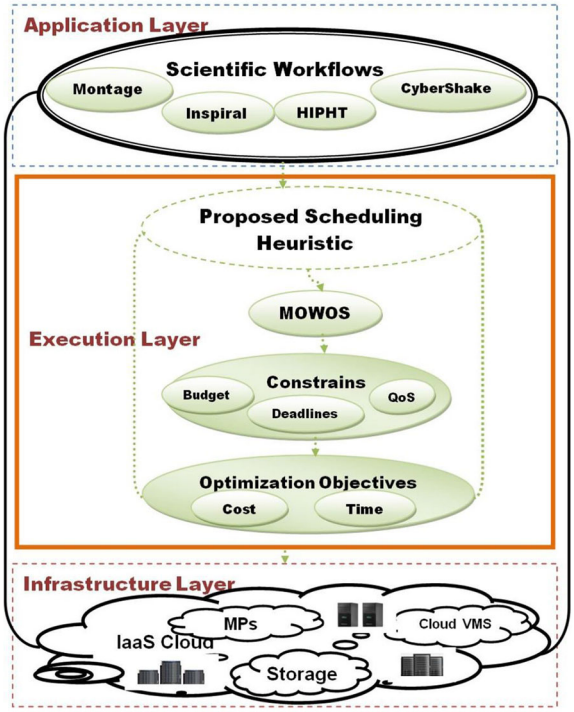

Fig. 3 Workflow execution model

the cost for executing workflow task wt_i on vm_k. At each step of the execution phase, expected completing time (ECT) of each task is generated and compared with the deadline of the task as specified by the user to determine whether the task can meet its deadline.

Expected Completion Time of workflow task wt_i on vm_k is denoted as $ECT(wt_i, vm_k)$, which can be computed by using Eq. 7 [46].

$$ECT = ET + loadvm_k \tag{7}$$

Where ET is execution time of workflow task wt_i on vm_k and $loadvm_k$ is the workload of vm_k at a given time.

Execution Time of workflow task wt_i on vm_k is denoted as $ET(wt_i, vm_k)$, and it can be calculated using Eq. 8.

$$ET = \frac{TL}{MIPS_{vmk}} \tag{8}$$

Where TL is the task length, MIPSvm_k is the Million instruction per seconds of vm_k.

3 **Infrastructure Layer**: Cloud provides software which is available via a third-party over the internet referred as software as a service (SaaS); services such as storage, networking, and virtualization known as infrastructure as a service (IaaS) and hardware and software tools available over the internet which is commonly referred to as platform as a Service (PaaS) [47]. The infrastructure layer which refers to as IaaS cloud provides services such as storage, networking, and virtualization services needed to support cloud computing to function. Our proposed method is making use of the services provided by the IaaS cloud such as storage, to store workflow applications, memory to process the applications, Physical Machines (PMs) to configure VMs to execute cloud users' requests.

Proposed algorithm: multi-objective workflow optimization strategy (MOWOS)

We Began by making the following assumptions:

1 All workflow tasks submitted can be split into sub-tasks.
2 Deadlines of workflows are known on arrival.
3 Every large workflow task, when split can meet its deadline.
4 Provisioned resources are available from the starting of the workflow exaction to its end.
5 The VMs workload have no affect on the tasks splitting process

In this section, we present the proposed MOWOS algorithm to optimize multi-objective. Multi-objective workflow optimization involves when two conflicting, but yet important scheduling objectives such as execution cost and execution makespan are optimized concurrently in the cloud computing environment. After extensive literature, it was revealed that most of the current heuristic algorithms are not robust in optimizing conflicting objectives such as execution cost and execution makespan simultaneously. Taking this into consideration, we extended our previous work (COHA) and presented a novel scheduling heuristic called Multi-Objective Workflow Optimization Strategy (MOWOS) for the purpose of reducing the system execution cost, time and fully utilize the resources, while ensuring all tasks meet their deadlines. The proposed method consists of three (3) sub-algorithms, namely: Task Splitting Algorithm, Minimum VM (MinVM) selection algorithm, and Maximum VM (MaxVM) selection algorithm. The pseudo-code of the proposed MOWOS is presented in algorithm 1. The algorithm starts by identifying all the arrived workflow tasks in the wt_{queue} with a set of workflow tasks Wt=(w_{t1}, w_{t2},w_{t3} w_{t4} ..... w_{tn}) with their corresponding execution lengths ($EL_{0.5}$, EL_1, $EL_{1.5}$ EL_2 .....EL_n) in a wt-$_{queue}$. These tasks need to be executed on cloud resources (VMs). In the proposed algorithm, we assumed there are only two sizes of VMs, that is VM_{Max} and VM_{Min}. We considered the set of VMs as (VM_i, VM_j, VM_k.....VM_n) with their corresponding sizes as (VM_{Max}, VM_{Min} ..... VM_{xx}). When a task arrived at the queue, MOWOS first calculates the expected completing time (ECT) of each workflow task in the wt_{queue}. Then, a new queue call a deadline queue (Dl_{queue}) is created and all the arrived tasks in the wt_{queue} are re-queued in the Dl_{queue} based on their user-specified deadlines. In the next step, the ECT of each workflow task is compared with its user-specified deadline. And if for example, an ECT of wt_i on VM_k is grater than its deadline, then the algorithm will apply the split method in algorithm 2 to split the task into sub-tasks and employ the minimum VM selection method (MinVM) to map the split tasks. MinVM selection method is introduced to map small tasks on a lower cost VMs. This is done to avoid scheduling small tasks on VMs with higher MIPS (high-cost VMs) to reduce the cost of execution. On the other hand, if the ECT of wt_i on VM_k is equal to its deadline, the MaxVM selection method (VM with a higher MIPS) will be deployed to execute the task faster. The MaxVM selection method is intended to map all workflow tasks that have equal ECT to their deadlines on a higher VMs to reduce makespan. Detail description of the sub-algorithms are explained herein:

Tasks splitting algorithm:

Reducing execution cost and execution makespan in the cloud computing environment, is an important issue for cloud service providers. If cloud service providers cannot reduce the cost and time of workflow scheduling, it may

Algorithm 1: MOWOS

Input : Given a DAG, G=(W,E), where w has n workflow tasks (wt_1, wt_2, tw_3,.....wt_n)
Output:

```
wt-queue-list = (wt1, wt2, tw3,.....wtn)
VMList = (vm1, vm2, ..... vmn)
while wt_queue ≠ ϕ do
    compute ECT of each wt using equation 8
    Create Dlqueue and queue the tasks based on their deadlines
    foreach ECT of wti on vmk do
        Compare the ECT of wti on vmk to its deadlines
        if the ECTwti > Dlwti then
            Apply the 'Splitting Algorithm
            Apply MinVM Algorithm to map the splits tasks and tasks with ECT<Dl
        else
            if ECTwti = Dlwti
            Apply MaxVM Algorithm
            Update Dlqueue
            while Dlqueue ≠ ϕ do
                Repeat step 7 to step 14
            end
            if Dlqueue is empty
        end
    end
end
```

with a set of workflow tasks Wt=(wt_1, wt_2,wt_3 wt_4 ..... wt_n). Afterward, the deadline for each task is compared with its ECT, and if the ECT of wt_i on vm_k is greater than its deadline, the task will be split into sub-tasks and sent to the ready queue for mapping decision. The algorithm terminates when all the large workflow tasks are successfully split.

MaxVM selection algorithm

The focus of most scheduling policies is to reduce the waiting time of tasks to allow them to meet their user-specified deadlines and thereby minimizes the makespan. Given this rationale, we introduce a method called maximum VM selection method (MaxVM selection) to map large tasks effectively on VMs to further reduced makespan and increase scheduling efficiency. Given a set of VMs as (VM_l, VM_m, VM_n.....VM_k) with their corresponding sizes as (VM_{Max}, VM_{Min} ..... VM_{xx}), the VMs are sorted in descending order and queued based on the execution speeds or sizes. Then the MaxVM selection method identifies VM in the VM queue with a higher execution capacity to execute the tasks that their ECTs are equal to their deadlines. The objective of this method is to help reduce the waiting time of workflow tasks with maximum execution time and thereby reduce the total execution makespan.

lead to customer dissatisfaction, which will consequently affect the profit margin of the service providers. The easy way to reduce the execution cost and time of workflow tasks, is to ensure workflows do not miss their deadlines. Scheduling large workflows tasks onto cloud resources delays the scheduling processes, thus making some of the workflows, to miss their deadlines. On the contrary, splitting large workflows tasks into sub-tasks, reduces the scheduling length of workflow tasks, thereby allowing every workflow task to meet their deadlines. Researchers like [48, 49] have proved that scheduling small size of workflow tasks on cloud resources provides better execution makespan and cost than mapping large workflow tasks on cloud resources. For this reason, we presented tasks splitting algorithm to split tasks that are likely not to meet their user-specified deadlines into sub-tasks, to reduce their scheduling lengths. This helps in: (i) given effective estimates in execution cost and time, (ii) identifying and fixing bottlenecks easily, and (iii) saving data transfer time. The pseudo-code of the tasks splitting algorithm is in algorithm 2. The algorithm starts by identifying all the arrived workflow tasks in the workflow tasks queue

Algorithm 2: Tasks Splitting Algorithm

Input : Given a DAG, G=(W,E), where w has n workflow tasks (wt_1, wt_2, tw_3,.....wt_n).
Output:

```
Begin
For wt-queue = (wt1, wt2, tw3,.....wtn)
Compare the ECT of wti to its deadline
if the ECTwti > Dlwti then
    Split the wti into sub-tasks
    Add the split workflow task to the wtqueue(wt1, wt2, wt1+1 ....wtn)
else
    if ECTwti =< Dlwti
    Put the task in the wtqueue for execution
    Update the wtqueue
    while wtqueue is not empty do
        Repeat step 4 to 10
    end
    if wtqueue is empty
end
```

Algorithm 3: MaxVM Selection

Input : List of VMs (vm_j, vm_K ..... vm_n) with different execution capacities

Output:

```
1  Begin
2  wt-queue = (wt1, wt2, tw3,.....wtn)
3  VM queue = (vmj, vmK ..... vmn)
4  foreach VM in the VMqueue do
5  |  get the MIPS of each VM
6  |  if vm.getMips > maxVM. getMips then
7  |  |  maxVM = VM;
8  |  end
9  |  return maxVM;
10 end
```

MinVM selection algorithm:

After splitting the large workflow tasks in algorithm 2, the tasks have to be mapped onto VMs for execution. When mapping tasks onto VMs for execution, certain decisions have to be made based on the objectives of the algorithm. In this research, the objective of the algorithm is to schedule a task for a reduced execution makespan and cost. For this reason, we introduce the MinVM selection method to map all the tasks that their ECTs are less than their user-specified deadlines. This is done to avoid mapping smaller workflow tasks on VMs with higher execution capacities that comes with higher cost and may lead to an increase in execution cost. The algorithm begins by identifying the MIPS of VMs (line 6), which is used to determine the VMs with lower MIPS (cheaper VMs) in the VM list (line 7) and then migrates all the tasks with short execution length onto the cheaper VMs to maximize profit. The rationale is that, as tasks are split in algorithm 2, and are less than their deadlines, low-cost VMs can easily execute them without delays. Table 1 presents the abbreviations and their definitions used throughout the paper.

Algorithm 4: MinVM Selection

Input : List of VMs (vm_j, vm_K ..... vm_n) with different execution capacities

Output:

```
1  Begin
2  VMqueue = (vmj, vmK ..... vmn)
3  VM Size (vmMax, vmMin ..... vmxx).
4  minVm = getVmSize
5  foreach vm in vmAllocation.keyset do
6  |  get the MIPS of each VM
7  |  if MIPS of vmk in the VMAllocation.keyset <
   |  MaxVM then
8  |  |  minVm = vm;
9  |  end
10 |  return minVm
11 end
```

Time complexity analysis

The overall time complexity of the proposed MOWOS algorithm is based on execution makespan and execution cost complexity, which depends on the number of workflow tasks and the size of VMs. It starts with algorithm 1 to calculate the ECT of all workflow tasks. Moreover, its loop at line 7 to generate a schedule by comparing the ECT of workflow tasks to their deadlines. In total, it takes $O(n + l)$ where n and l are the number of tasks and tasks length respectively.

Algorithm 2 iterate through to determine the size of each task and their deadlines to be able to identify the large tasks for splitting. The time complexity for task splitting depends on the size of the tasks and its deadline which is given as $O(s + d)$ where s is the size of the workflow task and d is the deadline of the workflow task in a workflow queue. After splitting a task, we search for maxVMs

Table 1 Abbreviations and their descriptions

Abbreviations	Description
Wt	Workflow task
n	Number of tasks
k	Number of VM types
ECT	Expected completion time
Dl	Deadline
Dl_{queue}	Deadline queue
wt_{queue}	Workflow tasks queue
MinVM	Minimum virtual machine
MaxVM	Maximum virtual machine
EL	Execution length
ET	Execution time
ECT	Expected completing time
loadVMj	The workflows on virtual machine j
$pre_{(wti)}$	Predecessor workflow task i
$succ_{(wtj)}$	Successor workflow task j
$wt_{entry)}$	Entry workflow tasks
wt_{exit}	Exit workflow tasks
MIPS	Million instructions per second
getMips	Get million instructions per second of Vm
getVmSize	Get the size of each VM (MIPS)
vmAllocation.keyset	Stores VM ID and Allocations
TL	Tasks length
$MIPS_{VMk}$	Million instruction per seconds of virtual machine K

in algorithm 3 and minVMs in algorithm 4 for allocation. The time complexity of algorithm 3 and algorithm 4 are the same. The time complexity for allocating tasks onto VMs is $O(n \times m)$. The overall complexity of MOWOS is given as $O((n + l) + (s + d) + (n \times m))$.

Performance evaluation

In this section, we present a set of extensive simulation experiments using different workflow inputs aim at evaluating the performance and contributions of the proposed MOWOS algorithm. Three different experiments are conducted in this section. In the first experiment, we evaluated the performance of the proposed algorithm in execution cost. The second phase of the experiment evaluates the performance of scheduling efficiency (makespan) in the cloud computing environment and in the third phase, we evaluated the resource utilization rate of the algorithm and compared the results with existing state-of-art workflow scheduling algorithms such as HSLJF [46] and SECURE [50]. This research adopted a simulation-based approach because, it offers cloud users the opportunity to pre-test the cloud services to determine their performance before they are made available to users in the real clouds [51]. This section consists of three sub-sections, namely; Experimental Setup, Workflow Structure, and Results and Analysis.

Experimental setup:

The focus of the experiment is to optimize execution cost, execution makespan and resource utilization, while meeting deadlines in Infrastructure as a Service (IaaS) Cloud. Cloud computing is viewed as a dynamic environment that makes it challenging to run large scale virtualized applications directly on the cloud data center [52]. We implemented the proposed method in a workflowSim simulator [53]. It is a java based simulation environment with the ability to model and simulates cloud scheduling algorithms [10]. Workflowsim was extended from CloudSim [54] to support the modeling of workflow DAGs in the cloud computing environment. We considered four different real-world benchmark workflows; Montage, CyberShake, SIPHT, and Ligo Inspiral as used in [55–58]. We grouped the workload of each of the workflows into four sizes including; small, medium, large, and extra-large as shown in Table 2. The groupings are based on standard benchmark-setting of real scientific workflow applications which have been practically used by different researchers to model and evaluate the performance of workflow scheduling algorithms similar to the work in [1, 13, 59–61]. Choosing different task sizes will afford the researcher the opportunity to measure the performance of the proposed method in different workloads. Also, in the experiment, we consider only four VMs with different configurations. The simulation environment is set as follows: Average bandwidth between resources are fixed at 20 MBps according to [62, 63], which is the approximate average bandwidth setting offered in Amazon web services [2, 64], the processing capacity of vCPU of each VM is measured in Million Instruction Per Second (MIPS) as in [65], the task lengths is set in Million Instruction (MI) according to [55]. The VM configurations, cost, and processing capacities are modeled based on the Amazon EC2 IaaS cloud offering as proposed by Ostermann et al. [41, 66]. Detail descriptions of the four VMs deployed are specified in Table 3. Specifically, the simulation experiments are conducted on a PC processor Intel (R) Core i5 1.6 GHz, 8 GB RAM using Windows 10.

Table 2 Types of workflows and their sizes

Workflows	Size of workflows			
	Small	Medium	Large	Extra-large
Montage	25	50	100	1000
CyberShake and Sipht	30	60	100	1000
Logo Inspiral	30	50	100	1000

Workflow structure

The simulation process was conducted using four different scientific workflows generated by the Pegasus workflow generator [67] such as Montage, CyberShake, SIPHT, and Ligo Inspiral. These workflows are from different scientific domains and come in large data sets that are structured differently [5]. The Fig. 4 symbolically represent the topological structures of scientific workflows. Detail descriptions of these workflows are presented in [67].

- Montage workflow is an astronomical application created by the National Aeronautics and Space Administration/Infrared Processing and Analysis Center. It is used for the construction of large mosaics of the sky. Montage application can be re-projected into input images for the correct orientation while maintaining the background emission level constants in all images [68]. Montage tasks are data-intensive and there do not require larger processing capacity to process[2].
- CyberShake is used in earthquake science to epitomize earthquake hazards by generating synthetic seismograms [23]. This is done for easy identification of

Table 3 VM types and pricing model

Virtual machines	Processing capacities (MIPS)	Cost
Micro	250	0.15
Small	500	0.3
Medium	1000	0.6
Large	2000	0.9

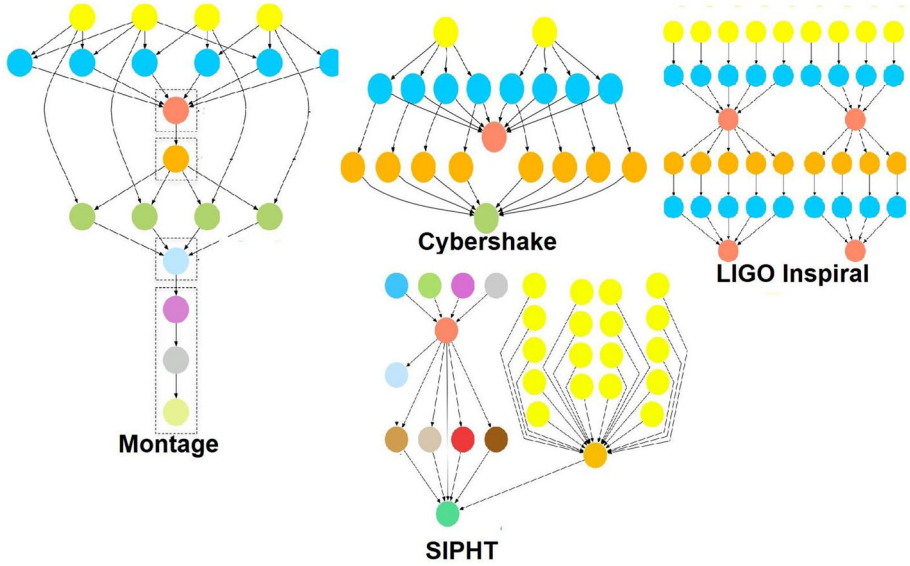

Fig. 4 Structure of scientific workflows used [67]

earth radiance and the production of accurate and reliable environmental estimates. CyberShake is a data-intensive workflow application, that requires a higher processing system with large memory to execute.

- LIGO Inspiral is the largest gravitational wave observatory in the universe. It is used in gravitational physics to exploit the physical properties of light and space to produce gravitational activities of the earth. Ligo Inspiral is a CPU intensive task and will need a large memory to process.
- SIPHT workflow application is from the bioinformatics project at Harvard which is used to automate the identification of RNAs (sRNAs) encoding genes for samples of bacterial in the National Center for Biotechnology Information database [60].

Results and analysis:

This section highlights the results obtained from a proposed MOWOS algorithm with other two existing state-of-the-art scheduling algorithms - the HSLJF and SECURE. In order to examine the performance of the proposed algorithm over the other two algorithms, the following three performance matrices were used;

1 The execution cost,
2 The execution makespan,
3 The resource utilization.

The Execution Cost: This is the budgeted total cost to get the workflow schedule on the cloud resources. This cost includes processing cost, the cost of transferring input and output files, etc. The execution cost results using the Montage workflow for different workloads such as 25, 50, 100, and 1000 are compared in Fig. 5. The three algorithms are compared to determine the best cost optimizer. In using the MOWOS algorithm, a lesser execution cost is achieved for every task size of montage. In other words, it is cheaper to generate a schedule with the proposed MOWOS algorithm than HSLJF and SECURE algorithms. Among the four workloads of montage, the SECURE algorithm produced the highest execution cost, because, it is not able to distribute workflows evenly on all the deployed resources.

The Fig. 6, illustrates the performance of various algorithms in terms of execution cost, using the Cybershake workflow on different workloads. The result shows that the execution cost increases with the number of workloads, for each of the evaluated algorithms. From Fig. 6, the proposed MOWOS algorithm has more advantages in reducing execution cost than HSLJF and SECURE algorithms. This is because the proposed method assigns workflow tasks to resources by considering both the workload and the resources capacity to handle the workload.

The SIHPT and Inspiral workflows (as in Figs. 7 and 8) were the next set of analysis to compare the execution cost among the MOWOS, HSLJF and SECURE algorithms. In comparison with the other algorithms, when MOWOS is used to generate a schedule, the rate of decrease in the execution cost is higher as the number of workload increases, as compared to HSLJF and SECURE algorithms. Quintessentially, an increase in the number of workloads

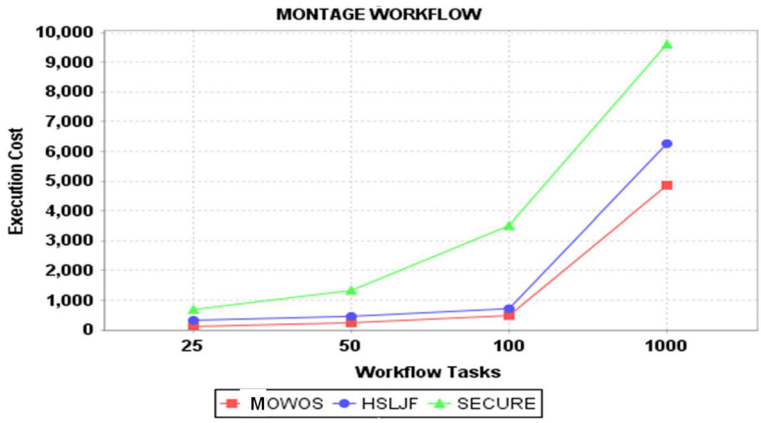

Fig. 5 Comparison of execution cost using Montage workflow

increases the number of split tasks. When more tasks are split into sub-tasks, it increases the scheduling efficiency, thereby, reducing the execution cost of workflows. Similarly, the SECURE algorithm is able to improve its performance significantly, by narrowing the execution cost gap in SIPHT workflow, and outperforms the HSLJF algorithm in the Inspiral workflow.

The four workflows results for the three workloads indicate that, it cost less to generate a schedule using the proposed MOWOS algorithm than the benchmarked algorithms. When the workload is increased, the overall performance of MOWOS gets much better, as compared to HSLJF and SECURE algorithms. The improvement in the proposed algorithm is that, the proposed algorithm employs VM selection mechanisms, thus making it easy to allocate tasks properly on VMs. Since HSLJF and SECURE do not use VM selection strategies like minVM and maxVM selection, it maps workflow tasks onto VMs without considering the capabilities of VMs. It is also obvious from the illustrations that, the cost of execution increases steadily, when large and extra-large workflows tasks are used. This is because, the cost is determined based on the number of computation and so, an increase in the number of workflows tasks (from 100 to 1000) will lead to a corresponding increase in the cost of execution.

Execution Makespan: The results of makespan for scheduling different workflow tasks onto a range of VMs is presented in Fig. 9. It shows the results of makespan using the Montage workflow with different workloads. The makespan is the total running time of a resource during workflow scheduling. In comparing the performance of the three algorithms, the proposed MOWOS algorithm is efficient in distributing workflow tasks on VMs in all the four different workloads (25, 50, 100, and 1000). The MOWOS algorithm, therefore, achieved a significant reduction in makespan as compared to HSLJF and SECURE algorithms. This is because MOWOS algorithm is able to identify and map the large workflow

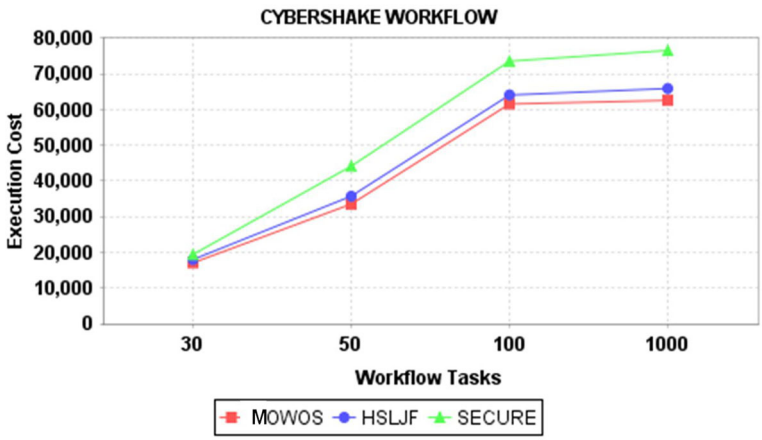

Fig. 6 Comparison of execution cost using CyberShake workflow

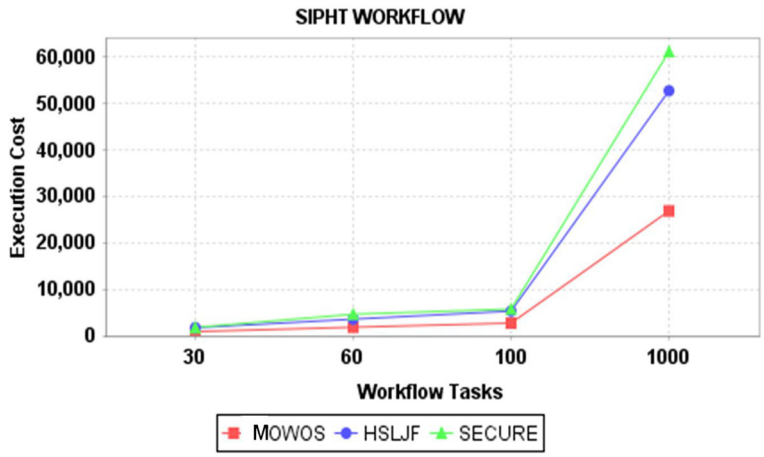

Fig. 7 Comparison of execution cost using SIPHT workflow

tasks onto MaxVMs and the small workflow tasks onto MinVMs to avoid overloading, since overloaded VM can slow down the execution process of workflow tasks. For example, in a workload of 25 workflow tasks, the proposed MOWOS took less than a minute to execute the 25 workflow tasks on VMs, while HSLJF and SECRE algorithms took more than a minute to execute the same workload.

Similarly, the Fig. 10 presents the results of makespan for CyberShake workflow, benchmarked under different workloads or task sizes (30, 50, 100, and 1000). It shows that the three algorithms are closely doing better in terms of execution makespan. It is however notices that, in terms of task execution time, the proposed MOWOS algorithm takes less time to execute workflows in all the four workloads, followed by the HSLJF algorithm and lastly the SECURE algorithm.

The results generally showed that the proposed algorithm consistently achieves lesser makespan values in both SIPHT and Inspiral workflows for all the workloads (Figs. 11 and 12). These results clearly demonstrate that, the proposed MOWOS algorithm generally generates the best workflows schedules than HSLJF and SECURE algorithms. The HSLJF algorithm also uses less time to execute tasks on VMs, as compared to SECURE algorithm. Workflow tasks are better performed in MOWOS algorithm because, it is able to split large tasks that allow every task to meet their deadlines.

Overall, the proposed MOWOS algorithm produces better makespan over HSLJF and SECURE algorithms. The method is therefore more effective when the range of workloads is increasing. So the more large tasks are split into sub-tasks, the higher the efficiency of workflow task execution, hence the resultant reduction of the

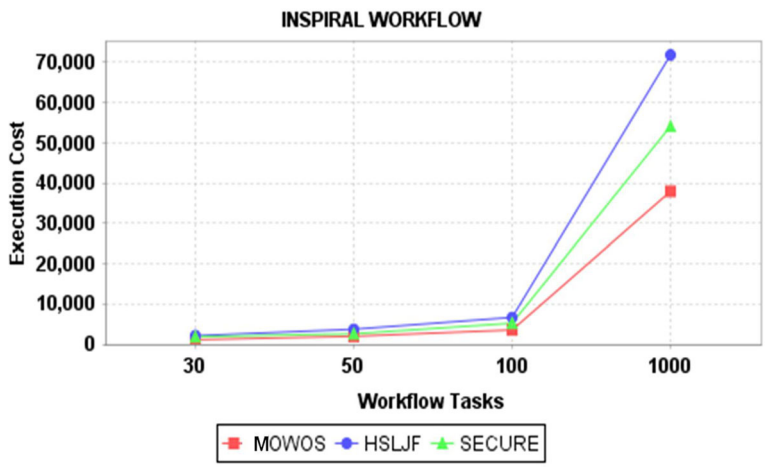

Fig. 8 Comparison of execution cost using inspiral workflow

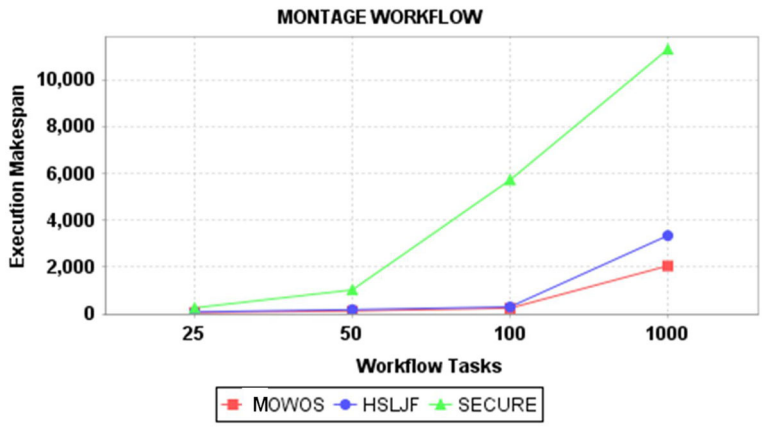

Fig. 9 Makespan results of montage workflow

execution makespan. However, by increasing the number of workflows tasks (from 100 to 1000), the execution time (makespan) increases, thus a reduction in the performance of the algorithms [69]. It is obvious from all the four workflows that, the three algorithms (MOWOS, HSLJF and SECURE) performs better with a lesser number of workflow tasks (from 25 to 50) as compared to the large and the extra-large workflows tasks (from 100 to 1000).

Resource Utilization: This refers to the practice of making the best use of cloud resources, by keeping the resources busy at all the stages of workflow execution. This is beneficial to the cloud service provider, because, providers maximize profit when resources are fully utilized. Any unused time slot of leased resources, is a cost to the provider, hence the need to ensure efficient use of cloud resources, to reduce cost and make profit. The utilization rate of cloud resources, by each of the three algorithms, were evaluated as in Figs. 13, 14, 15 and 16.

The figures illustrate the rate of resource utilization obtained by MOWOS, HSLJF, and SECURE algorithms. It is shown from the figures that for small workflows, the proposed MOWOS algorithm has a better performance than the existing HSLJF and SECURE algorithms by 10% and 13% respectively. In the medium workflows, the respective percentage improvement in the proposed MOWOS over HSLJF and SECURE is 12% and 13%. For the large workflows, the MOWOS algorithm has utilized the resources better than the existing HSLJF algorithm by 14% and SECURE algorithms by 24%. Lastly, for the extra-large workflows, the utilization improvement of the proposed MOWOS over the state-of-art-algorithms such as HSLJF is 18% and SECURE is 26%.

The overall trend of resource utilization rate is higher when large and extra-large workflow tasks are used. Conversely, the rate of increase in utilization falls when small and meduim workflows are scheduled. This is because any increase in the number of workflow tasks

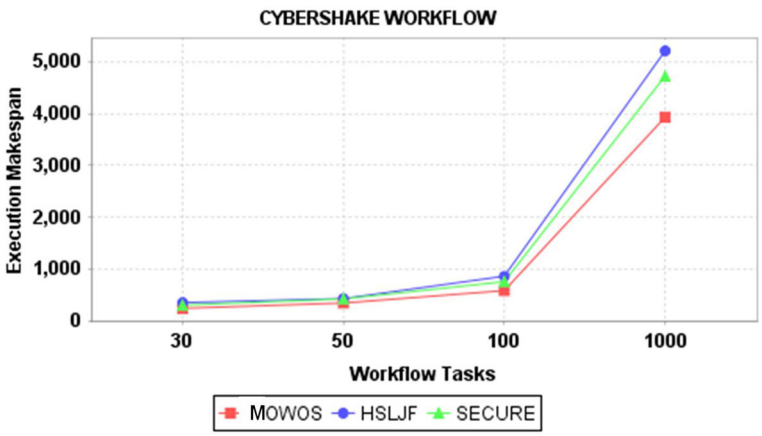

Fig. 10 Makespan results of CyberShake workflow

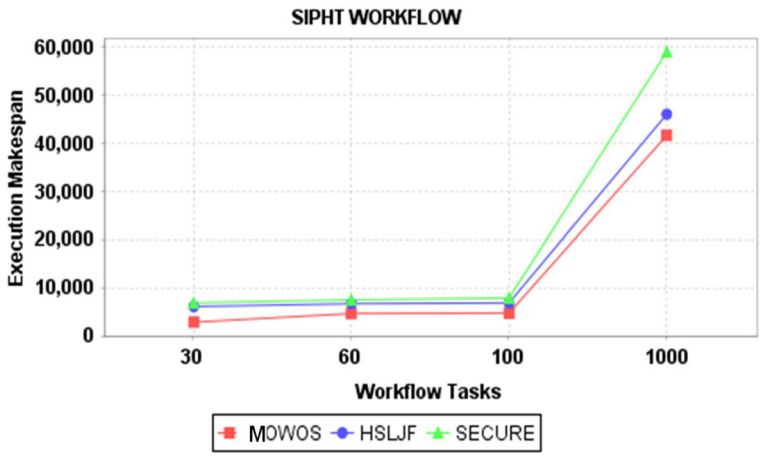

Fig. 11 Makespan results of SIPHT workflow

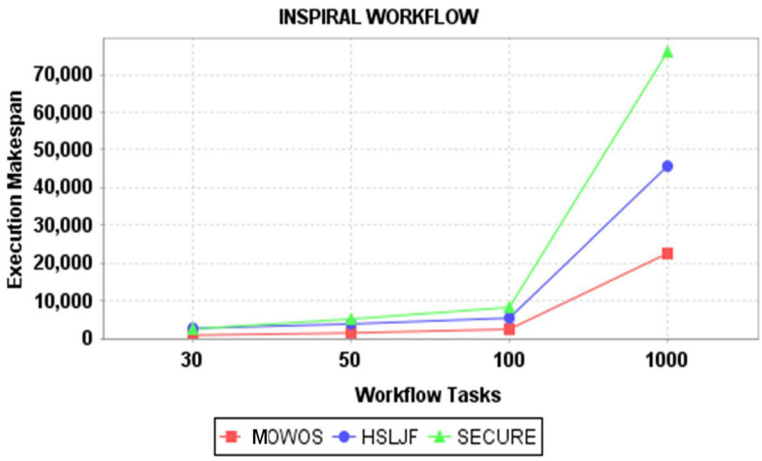

Fig. 12 Makespan results of Inspiral workflow

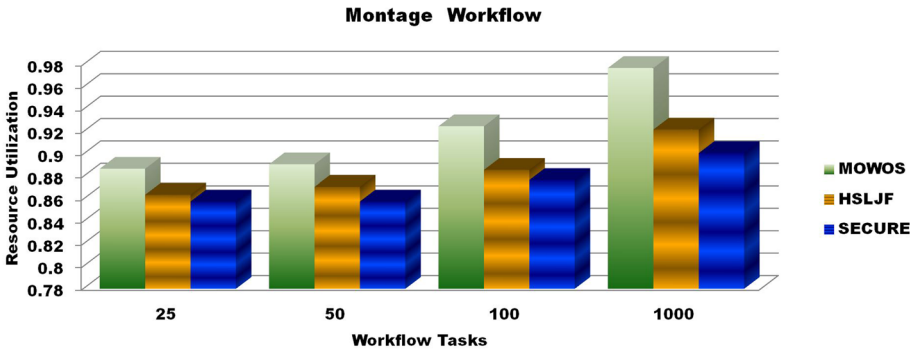

Fig. 13 Comparison of resource utilization using Montage workflow

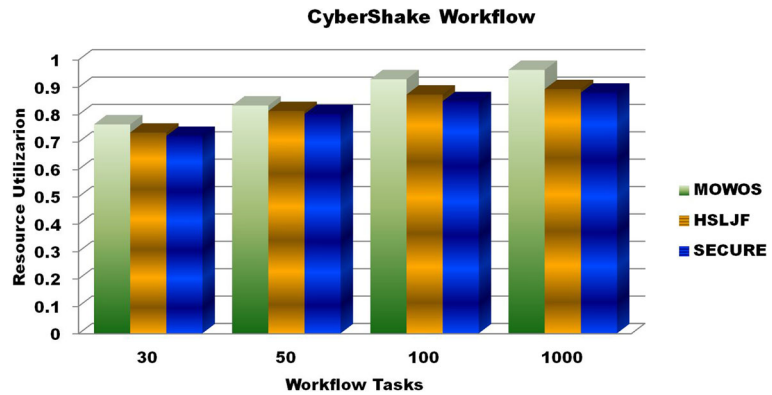

Fig. 14 Comparison of Resource Utilization using CyberShake workflow

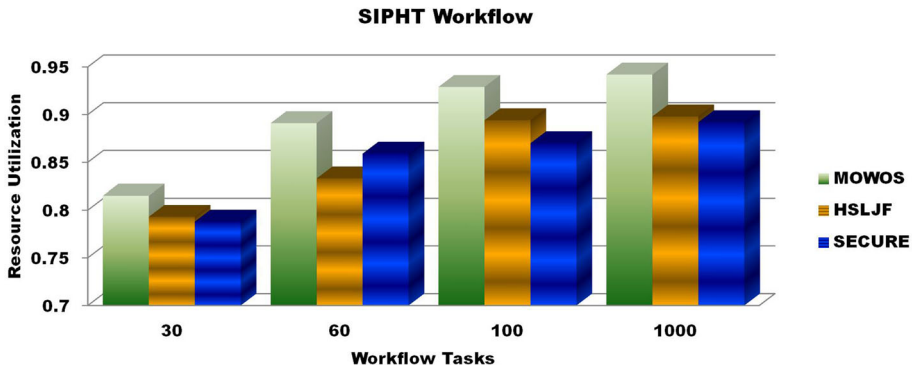

Fig. 15 Comparison of resource utilization using SIPHT workflow

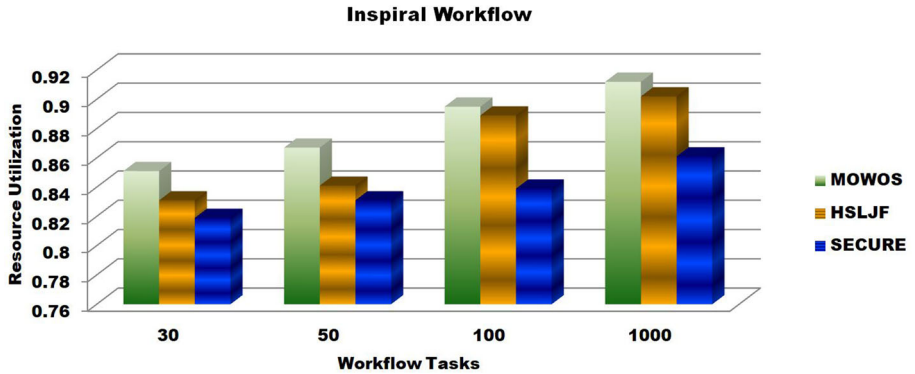

Fig. 16 Comparison of resource utilization using inspiral workflow

will lead to increased computation performed by each algorithm.

It is conclusive from the results analysis that, the proposed MOWOS algorithm has less execution cost, better execution makespan and utilizes the resources better than the existing HSLJF and SECURE algorithms.

Conclusion and future work

In this research, we investigated workflow scheduling problems in the cloud computing environment and presented a novel heuristic approach called Multi-Objective Workflow Optimization Strategy (MOWOS). The proposed algorithm aims to optimize execution cost, execution makespan and resource utilization, while allowing workflow tasks to complete before deadlines. The proposed algorithm consists of three sub-algorithms: a tasks splitting algorithm, a MaxVM selection algorithm, and a MinVM selection algorithm. The proposed method has been implemented and tested in the WorkflowSim simulator. We have compared the performance of the proposed algorithm with HSLJF and SECURE algorithms based on four well-known scientific workflows, including Montage, CyberShake, SIPHT, and LIGO Inspiral. The simulation results have shown that, the proposed MOWOS algorithm has less execution cost, better execution makespan and utilizes the resources better than the existing HSLJF and SECURE algorithms. The overall performance of the proposed algorithm increases significantly when compared to HSLJF and SECURE algorithms for large and extra-large workflow tasks while maintaining a slight improvement for small and medium workflow tasks.

In this paper, execution cost, execution makespan and resource utilization are considered as the only two optimization objectives. We will extend our work to consider energy consumption and load balancing in our future research, and also will provide more evidence to reasoning the assumptions made in this research.

Acknowledgements
This work was supported by the school of computer science, University College Dublin.

Authors' contributions
This research is part of Konjaang's Ph.D. thesis, which is supervised by Dr. Lina Xu. The author(s) read and approved the final manuscript.

Authors' information
School of Computer Science, University College Dublin, Dublin, Ireland

Funding
This work was supported by the School of Computer Science, University College Dublin, Ireland.

Availability of data and materials
The data required for the reproduction of these findings cannot be shared at this time. This data forms part of an ongoing PhD. study. It can be shared after the study is completed.

Competing interests
The authors declare that they have no competing interests.

Received: 17 August 2020 Accepted: 4 December 2020
Published online: 28 January 2021

References

1. Nasr AA, El-Bahnasawy NA, Attiya G, El-Sayed A (2019) Cost-effective algorithm for workflow scheduling in cloud computing under deadline constraint. Arab J Sci Eng 44(4):3765–3780
2. Sahni J, Vidyarthi DP (2015) A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment. IEEE Trans Cloud Comput 6(1):2–18
3. InfoTech L (2012) What is cloud computing. IBM J Res Dev 60(4):41–44
4. Rodriguez Sossa MA (2016) Resource provisioning and scheduling algorithms for scientific workflows in cloud computing environments. PhD thesis
5. Li Z, Ge J, Hu H, Song W, Hu H, Luo B (2015) Cost and energy aware scheduling algorithm for scientific workflows with deadline constraint in clouds. IEEE Trans Serv Comput 11(4):713–726
6. Chawla Y, Bhonsle M (2012) A study on scheduling methods in cloud computing. Int J Emerg Trends Technol Comput Sc (IJETTCS) 1(3):12–17
7. Masdari M, ValiKardan S, Shahi Z, Azar SI (2016) Towards workflow scheduling in cloud computing: a comprehensive analysis. J Netw Comput Appl 66:64–82
8. Zhou X, Zhang G, Sun J, Zhou J, Wei T, Hu S (2019) Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based heft. Futur Gener Comput Syst 93:278–289
9. Rodriguez MA, Buyya R (2017) Budget-driven scheduling of scientific workflows in iaas clouds with fine-grained billing periods. ACM Trans Auton Adapt Syst (TAAS) 12(2):1–22
10. Anwar N, Deng H (2018) Elastic scheduling of scientific workflows under deadline constraints in cloud computing environments. Futur Int 10(1):5
11. Zhao Y, Fei X, Raicu I, Lu S (2011) Opportunities and challenges in running scientific workflows on the cloud. In: 2011 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery. IEEE. pp 455–462. https://doi.org/10.1109/cyberc.2011.80
12. Ismayilov G, Topcuoglu HR (2020) Neural network based multi-objective evolutionary algorithm for dynamic workflow scheduling in cloud computing. Futur Gener Comput Syst 102:307–322
13. Manasrah AM, Ba Ali H (2018) Workflow scheduling using hybrid ga-pso algorithm in cloud computing. Wirel Commun Mob Comput 2018. https://doi.org/10.1155/2018/1934784
14. Yassir S, Mostapha Z, Claude T (2017) Workflow scheduling issues and techniques in cloud computing: A systematic literature review. In: International Conference of Cloud Computing Technologies and Applications. Springer. pp 241–263. https://doi.org/10.1007/978-3-319-97719-5_16
15. Das I, Dennis JE (1997) A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems. Struct Optim 14(1):63–69
16. Qin Y, Wang H, Yi S, Li X, Zhai L (2020) An energy-aware scheduling algorithm for budget-constrained scientific workflows based on multi-objective reinforcement learning. J Supercomput 76(1):455–480
17. Konjaang JK, Xu L (2020) Cost optimised heuristic algorithm (coha) for scientific workflow scheduling in iaas cloud environment. In: 2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing,(HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS). IEEE Computer Society. pp 162–168. https://doi.org/10.1109/bigdatasecurity-hpsc-ids49724.2020.00038
18. Yu S, Li K, Xu Y (2018) A dag task scheduling scheme on heterogeneous cluster systems using discrete iwo algorithm. J Comput Sci 26:307–317
19. Garey MR (1979) Computers and intractability: A guide to the theory of np-completeness. Rev Escola De Enfermagem Da USP 44(2):340
20. Awad A, El-Hefnawy N, Abdel_kader H (2015) Enhanced particle swarm optimization for task scheduling in cloud computing environments. Procedia Comput Sci 65:920–929
21. Rimal BP, Maier M (2016) Workflow scheduling in multi-tenant cloud computing environments. IEEE Trans Parallel Distrib Syst 28(1):290–304

22. Haidri RA, Katti CP, Saxena PC (2017) Cost effective deadline aware scheduling strategy for workflow applications on virtual machines in cloud computing. J King Saud Univ-Comput Inform Sci. https://doi.org/10.1016/j.jksuci.2017.10.009
23. Elsherbiny S, Eldaydamony E, Alrahmawy M, Reyad AE (2018) An extended intelligent water drops algorithm for workflow scheduling in cloud computing environment. Egypt Inform J 19(1):33–55
24. Kalra M, Singh S (2015) A review of metaheuristic scheduling techniques in cloud computing. Egypt Inform J 16(3):275–295
25. Jiang Y, Huang Z, Tsang DH (2016) Towards max-min fair resource allocation for stream big data analytics in shared clouds. IEEE Trans Big Data 4(1):130–137
26. Casas I, Taheri J, Ranjan R, Wang L, Zomaya AY (2018) Ga-eti: An enhanced genetic algorithm for the scheduling of scientific workflows in cloud environments. J Comput Sci 26:318–331
27. Liu G, Li J, Xu J (2013) An Improved Min-Min Algorithm in Cloud Computing. In: Du Z (ed). Proceedings of the 2012 International Conference of Modern Computer Science and Applications. Advances in Intelligent Systems and Computing, vol 191. Springer, Berlin. https://doi.org/10.1007/978-3-642-33030-8_8
28. Singh L, Singh S (2014) Deadline and cost based ant colony optimization algorithm for scheduling workflow applications in hybrid cloud. Int J Sci Eng Res 5(10):1417–1420
29. Zuo L, Shu L, Dong S, Zhu C, Hara T (2015) A multi-objective optimization scheduling method based on the ant colony algorithm in cloud computing. IEEE Access 3:2687–2699
30. Chaudhary N, Kalra M (2017) An improved harmony search algorithm with group technology model for scheduling workflows in cloud environment. In: 2017 4th IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics (UPCON). IEEE. pp 73–77. https://doi.org/10.1109/upcon.2017.8251025
31. Baxodirjonovich KN, Choe T-Y (2015) Dynamic Task Scheduling Algorithm based on Ant Colony Scheme. Int J Eng Technol 7(4):1163–1172
32. Li Y, Zhu Z, Wang Y (2018) Min-max-min: A heuristic scheduling algorithm for jobs across geo-distributed datacenters. In: 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS). IEEE. pp 1573–1574. https://doi.org/10.1109/icdcs.2018.00173
33. Ghumman NS, Kaur R (2015) Dynamic combination of improved max-min and ant colony algorithm for load balancing in cloud system. In: 2015 6th International Conference on Computing, Communication and Networking Technologies (ICCCNT). IEEE. pp 1–5. https://doi.org/10.1109/icccnt.2015.7395172
34. Fard HM, Prodan R, Barrionuevo JJD, Fahringer T (2012) A multi-objective approach for workflow scheduling in heterogeneous environments. In: 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012). IEEE. pp 300–309. https://doi.org/10.1109/ccgrid.2012.114
35. Sahni J, Vidyarthi DP (2018) A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment. IEEE Trans Cloud Comput 6(1):2–18
36. Amalarethinam DG, Beena TLA (2015) Customer facilitated cost-based scheduling (cfcsc) in cloud. Procedia Comput Sci 46:660–667
37. Adhikari M, Amgoth T, Srirama SN (2020) Multi-objective scheduling strategy for scientific workflows in cloud environment: A firefly-based approach. Appl Soft Comput:106411. https://doi.org/10.1016/j.asoc.2020.106411
38. da Silva RF, Casanova H, Orgerie A-C, Tanaka R, Deelman E, Suter F (2020) Characterizing, modeling, and accurately simulating power and energy consumption of i/o-intensive scientific workflows. J Comput Sci:101157. https://doi.org/10.1016/j.jocs.2020.101157
39. Kalyan Chakravarthi K, Shyamala L, Vaidehi V (2020) Budget aware scheduling algorithm for workflow applications in IaaS clouds. Clust Comput 23:3405–3419. https://doi.org/10.1007/s10586-020-03095-1
40. Arabnejad H, Barbosa JG (2017) Maximizing the completion rate of concurrent scientific applications under time and budget constraints. J Comput Sci 23:120–129
41. (2020) Amazon ec2 instance types [online]. http://aws.amazon.com/ec2/. Accessed 4 Jan 2020
42. Liu J, Pacitti E, Valduriez P, Mattoso M (2014) Parallelization of Scientific Workflows in the Cloud. [Research Report] RR-8565, INRIA. https://hal.inria.fr/hal-01024101v2
43. Deelman E, Vahi K, Juve G, Rynge M, Callaghan S, Maechling PJ, Mayani R, Chen W, Da Silva RF, Livny M, et al (2015) Pegasus, a workflow management system for science automation. Futur Gener Comput Syst 46:17–35
44. Deelman E, Singh G, Su M-H, Blythe J, Gil Y, Kesselman C, Mehta G, Vahi K, Berriman GB, Good J, et al (2005) Pegasus: A framework for mapping complex scientific workflows onto distributed systems. Sci Program 13(3):219–237
45. Juve G, Chervenak A, Deelman E, Bharathi S, Mehta G, Vahi K (2013) Characterizing and profiling scientific workflows. Futur Gener Comput Syst 29(3):682–692
46. Alworafi MA, Dhari A, El-Booz SA, Nasr AA, Arpitha A, Mallappa S (2019) An Enhanced Task Scheduling in Cloud Computing Based on Hybrid Approach. In: Nagabhushan P, Guru D, Shekar B, Kumar Y (eds). Data Analytics and Learning. Lecture Notes in Networks and Systems, vol 43. Springer, Singapore. https://doi.org/10.1007/978-981-13-2514-4_2
47. Savu L (2011) Cloud computing: Deployment models, delivery models, risks and research challenges. In: 2011 International Conference on Computer and Management (CAMAN). IEEE. pp 1–4. https://doi.org/10.1109/caman.2011.5778816
48. Dhanalakshmi M, Basu A (2014) Task scheduling techniques for minimizing energy consumption and response time in cloud computing. Int J Eng Res Technol (IJERT) 3(7):2278–0181
49. Al-Maytami BA, Fan P, Hussain A, Baker T, Liatsis P (2019) A task scheduling algorithm with improved makespan based on prediction of tasks computation time algorithm for cloud computing. IEEE Access 7:160916–160926
50. Singh H, Bhasin A, Kaveri P (2019) Secure: Efficient resource scheduling by swarm in cloud computing. J Discret Math Sci Cryptogr 22(2):127–137
51. Wang W-J, Chang Y-S, Lo W-T, Lee Y-K (2013) Adaptive scheduling for parallel tasks with qos satisfaction for hybrid cloud environments. J Supercomput 66(2):783–811
52. Choudhary A, Govil MC, Singh G, Awasthi LK, Pilli ES (2018) Task Clustering-Based Energy-Aware Workflow Scheduling in Cloud Environment. In: 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS). Exeter, United Kingdom. pp 968–973. https://doi.org/10.1109/HPCC/SmartCity/DSS.2018.00160
53. Chen W, Deelman E (2012) Workflowsim: A toolkit for simulating scientific workflows in distributed environments. In: 2012 IEEE 8th International Conference on E-Science. IEEE. pp 1–8. https://doi.org/10.1109/escience.2012.6404430
54. Calheiros RN, Ranjan R, Beloglazov A, De Rose CA, Buyya R (2011) Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw Pract Experience 41(1):23–50
55. Singh V, Gupta I, Jana PK (2019) An energy efficient algorithm for workflow scheduling in iaas cloud. J Grid Comput:1–20. https://doi.org/10.1007/s10723-019-09490-2
56. Gao Y, Zhang S, Zhou J (2019) A hybrid algorithm for multi-objective scientific workflow scheduling in iaas cloud. IEEE Access 7:125783–125795
57. Dubey K, Shams MY, Sharma S, Alarifi A, Amoon M, Nasr AA (2019) A management system for servicing multi-organizations on community cloud model in secure cloud environment. IEEE Access 7:159535–159546
58. Xie Y, Zhu Y, Wang Y, Cheng Y, Xu R, Sani AS, Yuan D, Yang Y (2019) A novel directional and non-local-convergent particle swarm optimization based workflow scheduling in cloud–edge environment. Futur Gener Comput Syst 97:361–378
59. Saeedi S, Khorsand R, Bidgoli SG, Ramezanpour M (2020) Improved many-objective particle swarm optimization algorithm for scientific workflow scheduling in cloud computing. Comput Ind Eng:106649. https://doi.org/10.1016/j.cie.2020.106649
60. Cotes-Ruiz IT, Prado RP, García-Galán S, Muñoz-Expósito JE, Ruiz-Reyes N (2017) Dynamic voltage frequency scaling simulator for real workflows energy-aware management in green cloud computing. PLoS ONE 12(1):0169803

61. Sun J, Yin L, Zou M, Zhang Y, Zhang T, Zhou J (2020) Makespan-minimization workflow scheduling for complex networks with social groups in edge computing. J Syst Archit:101799. https://doi.org/10.1016/j.sysarc.2020.101799
62. Arabnejad V, Bubendorfer K, Ng B (2018) Budget and deadline aware e-science workflow scheduling in clouds. IEEE Trans Parallel Distrib Syst 30(1):29–44
63. Mboula JEN, Kamla VC, Djamegni CT (2020) Cost-time trade-off efficient workflow scheduling in cloud. Simul Model Pract Theory:102107. https://doi.org/10.1016/j.simpat.2020.102107
64. Palankar MR, Iamnitchi A, Ripeanu M, Garfinkel S (2008) Amazon S3 for Science Grids: A Viable Solution? In: Proc. DADC '08: ACM Int'l Workshop Data-Aware Distributed Computing. pp 55–64
65. Rodriguez MA, Buyya R (2018) Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms. Futur Gener Comput Syst 79:739–750
66. Ostermann S, Iosup A, Yigitbasi N, Prodan R, Fahringer T, Epema D (2009) A performance analysis of ec2 cloud computing services for scientific computing. In: International Conference on Cloud Computing. Springer. pp 115–131. https://doi.org/10.1007/978-3-642-12636-9_9
67. Bharathi S, Chervenak A, Deelman E, Mehta G, Su M, Vahi K (2008) Characterization of scientific workflows. In: 2008 Third Workshop on Workflows in Support of Large-Scale Science, Austin. pp 1–10. https://doi.org/10.1109/WORKS.2008.4723958
68. Da Silva RF, Chen W, Juve G, Vahi K, Deelman E (2014) Community resources for enabling research in distributed scientific workflows. In: 2014 IEEE 10th International Conference on e-Science, vol. 1. IEEE. pp 177–184. https://doi.org/10.1109/escience.2014.44
69. Chana I, et al (2013) Bacterial foraging based hyper-heuristic for resource scheduling in grid computing. Futur Gener Comput Syst 29(3):751–762

Publisher's Note