International Journal on Advanced Science Engineering Information Technology

Vol.13 (2023) No. 1
ISSN: 2088-5334

Cost-effective and Low-complexity Non-constrained Workflow Scheduling for Cloud Computing Environment

Célestin Tshimanga Kamanga [a,*], Emmanuel Bugingo [b,c], Simon Ntumba Badibanga [a], Eugène Mbuyi Mukendi [a], Olivier Habimana [b]

[a] *University of Kinshasa, Mathematics and Computer Sciences, Kinshasa, 127, Democratic Republic of Congo*
[b] *University of Rwanda, KK 737 Street, Gikondo, Kigali, 4285, Rwanda*
[c] *University of Kigali, KG 7 Ave, Kigal, Rwanda*

Corresponding author: [*]celestin.tshimanga@unikin.ac.cd

***Abstract*— Cloud computing possesses the merit of being a faster and cost-effective platform in terms of executing scientific workflow applications. Scientific workflow applications are found in different domains, such as security, astronomy, science, etc. They are represented by complex sizes, which makes them computationally intensive. The main key to the successful execution of scientific workflow applications lies in task resource mapping. However, task-resource mapping in a cloud environment is classified as NP-complete. Finding a good schedule that satisfies users' quality of service requirements is still complicated. Even if different studies have been carried out to propose different algorithms that address this issue, there is still a big room for improvement. Some proposed algorithms focused on optimizing different objectives such as makespan, cost, and energy. Some of those studies fail to produce low-time complexity and low-runtime scientific workflow scheduling algorithms. In this paper, we proposed a non-constrained, low-runtime, and low-time-complexity scientific workflow scheduling algorithm for cost minimization. Since the proposed algorithm is a list scheduling algorithm, its key success is properly selecting computing resources and its operating CPU frequency for each task using the maximum cost difference and minimum cost-execution difference from the mean. Our algorithm achieves almost the same cost reduction results as some of the current states of the arts while it is still low complex and uses less run-time.**

***Keywords*— Workflow scheduling; resource management; difference from the mean; weighted sum difference; low complexity.**

Manuscript received 16 May 2022; revised 18 Aug. 2022; accepted 17 Sep. 2022. Date of publication 28 Feb. 2023.

I. INTRODUCTION

Cloud computing is a leading resource management system to run and manage high-performance applications from various natures. Such applications, including engineering, science, healthcare, finance, security, and business, require large amounts of data from diverse sources. It is straightforward that cloud applications require high-power computing resources, which are expensive to afford. Cloud computing services are provided by cloud providers and consumed by cloud clients [1].

Many users, individuals, and companies, now prefer to use cloud services over running their resources locally, as they provide high technical support with fewer implications for the users. To provide cloud services, a few concepts are combined. Most importantly, CPUs have been priced accordingly [2] and per application. Organizing its budget is still an ongoing concern in the cloud computing community, and providing the same cloud services with lower costs raises the bar. However, choosing a cost-effective configuration becomes even more difficult for users. The frequency required for a cost-effective configuration can vary in different scenarios, depending on the provider's pricing model and the characteristics of the application [3].

As a result, cloud computing utilizing virtualization technology has emerged as a new paradigm for large-scale distributed computing [4]. One of the basic challenges is to schedule a large set of heterogeneous tasks while maintaining load balancing between different heterogeneous systems to meet the requirement between the cloud users and providers [5]. Metrics to consider in this case include makespan and service fees which should be minimized and resources that should be highly used [6], [7].

Though cloud computing services have multiple advantages over traditional systems (i.e., hosting their own resources locally), some issues need to be addressed. One

question lies in the security of the used information being hosted by a third party. In other words, it is not easy to protect large amounts of data when it is stored in the cloud. Another issue is the recovery of lost data in contingency.

The issue considered in this work is that the resources required to manage and maintain user data in the cloud can be expensive. There are two principal factors considered when pricing cloud services. First, cloud providers aim to maximize profits, and cloud users need services of high quality at low prices. Second, cloud services are very competitive due to the high number of cloud providers. Other things that are taken into account to price cloud services include the cost of maintenance, age of services, and rate of depreciation.

This paper aims to balance cloud provider's profit against customer's satisfaction. A new approach is proposed to optimize the prices of services for both cloud providers and customers. There are still challenges in designing algorithms to meet user quality of service (QoS), minimize service costs, and reduce processing and response times. Cloud computing provides users with different computing resources and CPU configuration settings for each resource that they must choose to run workflow applications within their QoS requirements.

Researchers in the cloud community use workflow, a Directed Acyclic Graph (DAG), to model cloud-related problems. For the problem considered in this paper, DAG contains nodes with the following characteristics: expected execution time and the amount of data to be transferred between each pair of dependent nodes. Here, nodes are tasks that need to be executed.

While provisioned computational resources allow parallel execution of independent tasks and execution of computationally intensive scientific workflow applications, mapping workflow tasks to resources remains an NP-complete problem [8].

It even becomes more complex when other characteristics, such as resource configuration settings, have to be taken care of. Some service providers, such as ElasticHosts and cloud-sigma [9], charge the user based on the CPU frequency selected for each computing resource. Choosing a high CPU frequency for each task and allocating task resources appropriately will reduce the execution time of the workflow application but will increase the overall cost charged to the user. On the other hand, choosing the lowest CPU frequency can reduce the overall financial cost and increase the execution time.

However, the cost of minimum CPU frequency is not efficient because the total cost depends on the execution time. Therefore, choosing the minimum CPU frequency for each selected computing resource is not cost-effective [10], but it can be achieved by careful selection of the CPU frequency between minimum and maximum.

HEFT [11] is a well-known list-scheduling algorithm known for its low complexity. Despite the improvement and modification made for this algorithm, it is initial [12] purpose was to minimize the makespan of the workflow. Even if HEFT has been proposed over a few decades, it still carries the flag of low complexity.

Over the last few years, the problem of total monetary optimization has become the most difficult issue, resulting from the number of configuration settings the user can choose when performing each task in the workflow. Various studies have been conducted, and various algorithms have been proposed to find a solution to this problem [13].

These algorithms find the best schedule within the user's deadline, but response time and complexity are still high. In this article, we have proposed an unconstrained list scheduling-based workflow scheduling algorithm with low response time and complexity.

The proposed algorithm allows users to choose computational resources and their operational CPU frequencies to optimize the total economic cost of running workflow applications in a cloud computing environment. Two different scientific workflows (Montage and Inspiral) of different sizes (small, medium, and large) were used to evaluate the proposed algorithm through simulation. Considering the user's deadline, the results show that the proposed algorithm works well in a small workflow and with a small number of computing resources because it is within the deadline used. Considering the main objective (cost minimization), our previous work [2] has proposed three scheduling heuristics named WS-HEFT, MD-HEFT, and ED-HEFT, and evaluated them by simulation. In this new paper, the main contributions are listed as follows:

- we proposed a list scheduling-based workflow with low complexity and less response time.
- instead of overlapping mutations, the proposed algorithm selects computing resources and their operating CPU frequencies based on the difference between the average cost and execution time of each task.
- while the proposed algorithm is non-constrained, the simulation results have shown that the proposed algorithm participates even in deadline acceptance for any small-sized workflow type and a small number of computing resources.
- like all compared state-of-the-art algorithms, the proposed algorithm reduced the cost over HEFT. Additionally, in many cases, the proposed algorithm even reduced the cost over the compared state-of-the-art algorithms.

Workflow scheduling and resource provisioning are the hottest topics in cloud computing [14]. Different studies have been carried out, and workflow scheduling and resource provisioning algorithms have been proposed to address the mentioned problems and deal with different optimization objectives in workflow scheduling and resource provisioning.

Some scheduling algorithms focused on the minimization of the execution time of the whole workflow [15], [12], others concentrated on the total monetary minimization [16], [17], [18], while others concentrated on the optimization of the energy used by the computation resources during workflow execution [4], [19], [20]. Some studies proposed metaheuristics such as evolutionary algorithms and particle swarm optimization [21], [22] to tackle workflow scheduling problems.

However, those meta-heuristics do not converge so quickly. Recent studies are using deep learning techniques [23], [24], [25] for task scheduling and resource provisioning. In terms of resource provisioning, the most focused objective is the maximization of resource utilization [24], [25]. To minimize the makespan, HEFT [26] was proposed. In its nature, HEFT is a list-scheduling algorithm with low

complexity. Given workflow tasks and computing resources, HEFT arranges tasks in a list according to their upwards rank values and then schedules one after another to the computing resource capable of minimizing task's execution time while considering the communication time of dependent tasks.

This algorithm has been extended in various [12] ways to address different scheduling problems. Since HEFT manages to generate a fair schedule with low complexity, it has been widely employed in many other meta-heuristic algorithms [27], and in scheduling algorithms [10], [28], [29] to generate the initial schedule. Workflow scheduling can be based either on a single objective or a multi-objective. Most of the existing work has considered single objective programming, such as minimization of total monetary cost under different constraints like the deadline [30], [31]. However, real-world workflow scheduling sometimes requires more than one objective to satisfy the user's QoS requirement.

Taking cost as one of those objectives [32] has developed multi-objective models to deal with workflow scheduling problems in the cloud environment. Most works done on workflow scheduling have considered time used as the only feature to use, charging the monetary cost [33]. However, ElasticHosts and cloud sigma [9] have proven that it is possible to charge the user based on the CPU frequency used to execute each task on a given computing resource.

With this new schema, the problem that arises is the proper selection of computing resources and tuning its operating CPU frequency for each task so that the total monetary cost is minimized under a given constraint.

To provide the solution to the problem of selecting a cost-efficient CPU frequency configuration, Faragardi et al. [10] have assumed that the users are charged based on the CPU frequency allocated to each resource during the execution time of scientific workflow. Their works have considered the new pricing feature (CPU frequency). They have used a set of pricing models (linear, sublinear, and superlinear), which can be used to reduce costs. Their models and feature of consideration are reasonable for scientific workflow application processing.

In conjunction with CSFS-Max and CSFS-Min [34], HEFT has been used to help the user to choose the best CPU frequency configuration setting for each. Saeedizade et al. [35], employed HEFT to perform optimization of makespan while satisfying the user's QoS requirement(budget constraints). HEFT has also been employed [36] as the makespan-aware scheduler.

Note that the main objective of the algorithm proposed in this study [37] was to help users split a sum of frequency onto a fixed number of resources by giving each resource an identical frequency configuration so that the makespan can be reduced. CFMAX, CFMIN, and CCR aim to minimize the total monetary cost the user must pay when scheduling his/her workflow application in a cloud computing environment [16]. All of those algorithms employed HEFT to generate their initial schedule.

They remap the task to another computing resource and change the CPU frequency based on the initial schedule generated using HEFT. The algorithm [27] proposed employed HEFT for two reasons: task ranking technique and managing the deadline based on makespan. Considering this fact, the complexity of those algorithms increases because they have to wait for the schedule generated by HEFT before they start to produce their schedules. Moreover, this will also increase the run-time of those algorithms.

Contrarily to those algorithms, the algorithm proposed in this paper employs only from HEFT task ranking technique and does task resources mapping based on the maximum and minimum cost-execution difference values from the mean value. As the proposed algorithm is not constrained, it is straightforward; no task remapping is needed, which reduces run-time. Most of the existing works have concentrated on the optimization of different constraints related to workflow scheduling by considering both at the same time or one of them [38], [39], [40], [41].

Considering the budget constraint, proposed a bi-criteria priority particle swarm optimization (BPSO) algorithm to schedule workflow applications to cloud computing resources in a manner that optimizes both the monetary cost execution time related to scheduling user's workflow in a cloud environment. One-phase algorithm IC-PCP and two-phase PCP algorithm [42] employed a partial critical path algorithm and proposed polynomial time complexity suitable for large workflows. The objective of this work was to minimize the cost and execution of a workflow while still meeting the user's deadline.

Like this group of algorithms, our algorithm focused on minimizing cost. Contrarily, they are constrained, while our algorithm is not constrained. Unlike all the research above, the algorithm presented in this paper assumes the presence of different cloud computing resources with different CPU frequency configuration settings. The selected computing resources and their operating CPU frequencies determine the monetary cost the user has to pay.

This paper proposes a workflow scheduling algorithm that is less complex and has no run-time constraints. To achieve this, the algorithm uses the minimum and maximum cost and execution values generated by each CPU frequency configuration setting.

II. Materials and Method

This paper considered the problem of selecting a proper computing resource and tuning its CPU frequency so executing a task requires less monetary cost. Note that even if we considered an unconstrained system, the user's long execution time of the whole schedule is not acceptable. This section describes the considered application model, the parameter settings of the considered environment, and the cost model.

A. Computing Resources Model

In this paper, we described the cloud computing environment using a set of computing resources $CR = cr1, cr2, \ldots, cn$. Each of the CR configured with CPU frequency parameter settings in between maximum f_{max} and minimum f_{min} with a step CPU frequency f_{step} that is used to change the frequency from maximum to minimum frequency, as shown in Table II.

Each computing resource is provisioned for the whole execution time of the user's application. However, the user is only charged based on the CPU frequency allocated to each CR during the execution time of the task t_i that was mapped on

this CR. Moreover, no preemption is allowed, i.e., a CRp will be busy for the whole task being executed on it.

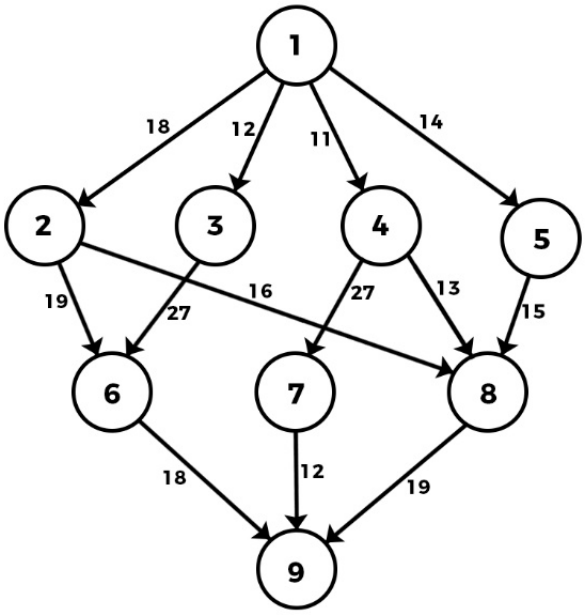

Fig. 1 Example workflow

TABLE I
EXPECTED EXECUTION TIME OF EACH WORKFLOW'S TASK

ID	CR1	CR2	CR3
1	30	12	16
2	27	21	72
3	4	36	6
4	21	6	20
5	20	16	81
6	28	6	48
7	35	14	7
8	5	48	30
9	48	2	64

TABLE II
EXAMPLE OF 3 CR'S CPU FREQUENCY PARAMETER SETTINGS

CR	fmax	fmin	fsep
1	3000	1000	100
2	2800	1400	200
3	2700	1800	300

B. Application Model

Commonly, scientific workflow applications can be modeled as a Directed Acyclic Graph (DAG) as illustrated in Fig. 1, W=(T,D), where T depicts a set of workflow's interdependent tasks T={t_1,t_2,t_3,…,t_n} and D depicts a set of intermediate data between each two dependent tasks D=$d_{(i,j)}$ where t_i is the predecessor (pred) of t_j which is the immediate successor (succ) of t_j. Additionally, pred(t_i) represent a set of all predecessor's task of t_j, and succ(t_j)represent a set of all successors task of t_i. A task without predecessor is known as entry task t_{entry}, while the task without successors is known as an exit task t_{exit}.

A simple representation of scientific workflow with 9 tasks is shown in Fig. 1. The example DAG have one $t_{entry=1}$ and one $t_{exit=9}$. The node represents the task, the number in the node represents the task's id, while the data on the edges represent the data to be transferred between the two dependent tasks $t_{(i,2)}$. A successor task t_j cannot start its execution before its predecessor t_i finishes its execution.

We assume the deterministic model of execution time and communication model. The tasks' expected execution time of sample DAG shown in Fig. 1 on three computing resources with maximum CPU frequency settings shown in Table II are presented in Table I.

The execution time of a task t_i on CPU frequency, which is not maximum, can be calculated as follows:

$$E_{t,f} = \beta \times \left(\frac{f_{max}}{f_{t,f\,max}}\right) \quad (1)$$

where $E_{(t,fmax)}$ is the expected execution time of task t_i at maximum CPU frequency f_{max} of computing resource cr_p as given in Table I: Expectation execution time of workflow shown in Fig. 1.

The parameter β indicate the impact of the CPU frequency on the execution time of the task. β is in range of 0 and 1. By default it is set to 0.4 (β=0.4).

We considered three pricing models: linear, super linear, and sublinear as in [2], [16], which were previously presented in [10]. Let suppose that $C_{(cr,f)}$ denotes the price charged per time unit's use of computing resource cr operating at CPU frequency f_{cr} $C_{(cr,fmin)}$ denotes the price of cr at minimum frequency f_{min}, and θr denotes the coefficient used for each cr to tune the changing rate of price according to frequency. Then $C_{(cr,f)}$ for linear pricing model is calculated as follows:

$$C_{(cr,f)} = Cr\left(\frac{f_{cr}-f_{min}}{f_{min}}\right)_{(cr,f_{min}} \quad (2)$$

and for super linear pricing model, $C_{(cr,f)}$ is calculated as follows:

$$Z = log\left(\frac{f_{cr}-f_{min}}{f_{min}}\right) \quad (3)$$

$$C_{(cr,f)} = Cr\left(\left(1+\frac{f_{cr}-f_{min}}{f_{min}} \times Z\right)\right)_{(cr,f_{min}} \quad (4)$$

while for sublinear pricing model, $C_{(cr,f)}$ can be calculated as follows:

$$C_{(cr,f)} = Cr\, log\left(1+\frac{f_{cr}-f_{min}}{f_{min}}\right)_{(cr,f_{min}} \quad (5)$$

Let also $TC_{(cr,f)}$ be the cost of executing task t_i to a computing resource cr operating at CPU frequency f and is calculated as follows:

$$TC_{(cr,f)} = C_{(cr,f)} \times E_{(t,f)} \quad (6)$$

C. The Proposed Algorithm

The first stage of our proposed algorithm is described in the figure below.

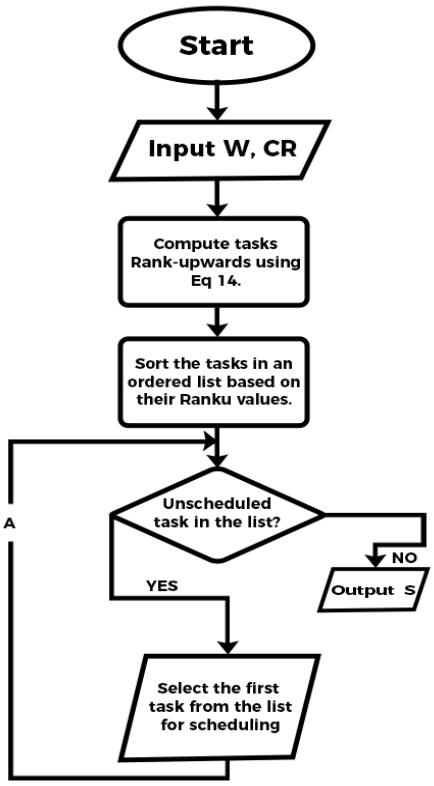

Fig. 2 Algorithm 1 of MCD and MCED

In case there is still an unscheduled task in the list, algorithm 1 will call algorithm 2 within Table III below, from point A.

TABLE III
ALGORITHM 2: SECOND STAGE OF MCD AND MCED

Step 1	: Create $EList = \phi$and $TCList = \phi$ Calculate $E_{(t,cr,f)}$ and $TC_{(t,cr,f)}$using Equation 1 and 6 respectively. $EList \leftarrow E_{(t,cr,f)}$,$TCList \leftarrow TC_{(t,cr,f)}$ Using EList and TCList Calculate the average AvE and AvTC. Switch Variants
Step 2	: Case (MCD) Find the proper CR cr and its selected frequency f using Equation 8. Break.
Step 3	: Case (MCED) Find the proper CR cr and its selected frequency f using Equation 13. Break.
Step 4	: Assign task t to the CR cr with frequency f that optimize $TC_{(t,cr,f)}$

Then, the total monetary cost of the $task_{cr}$ mapping can be calculated as follows:

$$TotC_s = \sum_{\forall(t,cr)\in S} TC_{(cr,f)} \quad (7)$$

where S is the schedule describing the mapping between tasks and computing resources as well as the operating CPU frequency of each computing resource for each task.

Based on the aforementioned models and assumptions, the main objective of this research is to propose a workflow scheduling algorithm that generates a schedule by appropriately tuning the CPU frequency for each task on a selected computing resource so that the total cost of executing user's workflow is minimized.

However, in this paper, we proposed an algorithm with two variants Minimum Cost Difference (MCD) and Maximum Cost-Execution Time Difference (MCED). The proposed algorithm uses Rank Upwards from HEFT and is calculated using Equation 14.

In this section, we provide a detailed description of proposed variants and discuss the results of the proposed algorithm compared to two more algorithms [43]. The proposed algorithm's pseudo-code is shown in the first algorithm, and the main method concerning variants are shown in the second algorithm.

1) Step by Step description: Normally, the proposed algorithm gets two inputs: The first input is a workflow W which consists of interdependent tasks with a deterministic model of execution time as shown in Table I (at maximum CPU frequency of the available CR) and communication time as shown on the edges in Fig. 1. The second input is a set of computing resources CR which also consists of several computing resources and their CPU frequency configuration settings as shown in Table II.

The output of the algorithm is a schedule S, specifying the $task_{cr}$ mapping with selected CPU frequency for each task scheduled on a given CR, and the generated E and C for each task. This algorithm consists of two main phases: Compute Rank upwards (Ranku) in the algorithm 1, and find the appropriate CR and its operating CPU frequency for each task. After getting the required input, the algorithm goes on with the computation of Rank upwards(Ranku) of all tasks by traversing W upward (starting from the exit task), and sorting the tasks in an ordered list based on their Ranku values.

After the calculation of the Ranku, each task in the list order can be selected based on its position in the list for the preparation of the execution (Step 1 of algorithm 2). We create two empty lists: where EList is used to hold all possible execution times of task t_i, TCList used to hold all possible execution monetary costs of task t_i.

For each computation resource in the given set with all its possible combinations of CPU frequency, we calculate $E_{(t,cr,f)}$ using Equation 1 and $TC_{(t,cr,f)}$ using Equation 6 and store those values in EList and TCList respectively. The next step is to calculate average execution time AvE and average execution cost AvTC.

$$MCD_{(cr,f)} = min(AvTC - TCList_i) \quad (8)$$

$$a = max(AvTC - TCList) \quad (9)$$

$$NCD_{(cr,f)} = w \times \left(\frac{a-(AvTC-TCList_i)}{a-min(AvTC-TCList)}\right) \quad (10)$$

$$b = max(AvE - EList) \quad (11)$$

$$NED_{(cr,f)} = w \times \left(\frac{b-(AvE-EList_i)}{b-min(EList)}\right) \quad (12)$$

$$MCED_{(cr,f)} = max\left(NCD_{(cr,f)} + NED_{(cr,f)}\right) \quad (13)$$

where w is the parameter used to control the criteria importance. In this paper, we assume that both execution time and execution cost have equal importance, which makes w=0.5.

Once the AvTC and AvE are found (Step 1 of algorithm 2), the next step is to calculate the difference between the average and each element in the list. If the variant is MCD (Step 2 of algorithm 2), we use the Equation 8, and the cr and f will be the ones used to get this minimum difference value from the mean. If the variant is MCED (Step 3 of algorithm 2), we use the Equation 13, and the proper cr and f will be the ones used to get them the maximum difference value from the mean. Before using Equation 13, execution values and cost values are firstly normalized: Normalized execution cost difference $NCD_{(cr,f)}$ using Equation 10 and normalized execution time difference $NED_{(cr,f)}$ using Equation 12.

To select the proper cr and f, task t_i looks only on its predecessors. Once the cr and its operating CPU frequency f is found, the task t_i will be scheduled for it, and the algorithm continues to the next task (Step 4 of algorithm 2). Note that after finding a proper cr with its associate CPU frequency f for task t_i, both EList and TCList are reset to prepare the next task $t_{(i+1)}$. The algorithm will return the schedule S and terminate when there is no remaining unscheduled task. S consists of the summation of the execution time of the tasks also known as makespan and total monetary.

From Fig. 1 , the rank upward is computed by traversing the task graph upward [44], starting from the exit task. For the exit task, the upward rank value is calculated by:

$$Ranku(n_{exit}) = \overline{w_{exit}} \quad (14)$$

Where $\overline{w_{exit}}$ is the average computation cost of task n_i and Ranku is the length of the critical path from task n_i to the exit task, including the computation cost of task n_i.

2) Time complexity: In this subsection, we analyze the time complexity of the two proposed variants. We let n represent the number of tasks made-up of a workflow, and m represent the number of possible combinations of CPU frequencies and computing resources that can exist according to the given parameter settings.

The time complexity of HEFT is known to be $O(n^2 \times m)$ and the performance of CFMAX is $O(n^2 \times (m + n))$ shown in [45]. Like HEFT, the proposed algorithm transforms the computing resources and their frequencies into a single unity, and then the search happens in a big size m than the one of HEFT. So the complexity of MCD is $O(n^2 \times m)$, while the one of MCED is $O((n^2 \times m) + m)$ where the second m comes from the search of maximum difference value using both cost and execution.

TABLE IV
CPU FREQUENCY PARAMETER SETTINGS

CR	f_{max}	f_{min}	f_{step}
1	4200	2100	300
2	2200	1200	200
3	3600	2400	300
4	3000	2000	200
5	4200	2100	300

TABLE V
WORKFLOWS SIZE

Workflow	Small	Medium	Large
Montage	25	100	1000
Inspiral	30	100	1000

TABLE VI
PRICING MODELS CONFIGURATION PARAMETERS

Pricing Model	Cmin ($)	θ ($)
Linear	9.24*10-8	3.30*10-8
SubLinear	2.78*10-8	1.2*10-8
SuperLinear	9.24*10-8	4.44*10-8

III. RESULTS AND DISCUSSION

This section describes in detail the environment used to develop this algorithm, the parameter settings considered in the algorithm, the type of workflow considered, the type of experiments we carried out, and then discusses the results found compared to other existing state-of-the-art algorithms.

A. Evaluation Settings.

The proposed algorithm was implemented and simulated using JAVA. Each considered computing resource CR can operate on different CPU frequencies between the maximum fmax and minimum fmin, and the change rate is governed by step frequency fset. Table IV demonstrates the considered CR and CPU frequency settings.

We considered the three pricing models: linear, super linear, and sublinear. More details about those pricing models can be found in [43]. Despite many workflow types, in this paper, we considered only two of them(MONTAGE and INSPIRAL [46]). Different from some of the current state-of-the-art, in this paper, we considered two workflows with different sizes, as shown in Table V.

We downloaded the DAX file for each workflow and size from the Pegasus workflow generator website [46]. Those files are in our simulation as the inputs to the proposed algorithm. DAX files contain the tasks' expected execution time expressed in Table I. It also contains data to be transmitted among every two dependent tasks. Besides, there are two parameters C_{min} and θused in Equations 2,4 and 5 whose values presented in Table VI, and were previously used in [43]. Those values are selected to estimate the monthly charges of ElasticHosts for the provisioning of CR.

B. Performance Impacted by the Workflow Size

To evaluate the performance of the proposed algorithm, we considered several CR in the range between 5 and 35 inclusive. Although our algorithm is not constrained, we used different deadline ratios dr to monitor the competing algorithms' deadlines and to which extent of the deadline our algorithm can be compared to those algorithms. We considered dr=7.

1) Small sized workflow: The cost results of Montage are substantially smaller than the cost results of Inspiral, based on the findings of a small-sized workflow. In comparison to HEFT, all of the algorithms had lower monetary costs. The MCD obtains lower cost values than CFMAX and CCR when the workflow is Montage. However, when the pricing model is Superlinear and Sublinear, MCED cost results are lower than CFMAX and CCR. Furthermore, when the pricing model is super linear, there is no discernible difference between the cost results given by the competing algorithms for all of the CR tested for both workflows. In all scenarios and workflows, it is also obvious that when the number of CR is 10, the algorithm generates smaller cost values. There is no

discernible difference between the cost results produced by our algorithms and those produced by CFMAX and CCR when the workflow is Inspiral. In terms of makespan, one can conclude that the makespan results of all the competing algorithms are within the deadline. The makespan results of HEFT are always lower than all other algorithms because this is a baseline algorithm that focuses on makespan optimization. When the pricing model is linear and workflows are Montage and Inspiral, MCED generates fewer makespan values than other algorithms. This case also appears in some cases (CR= [20 to 35]) when the pricing model is sublinear and workflow is Montage. Other results show that CFMAX and CCR have lower makespan than those generated by MCD and MCED.

2) Medium-size workflow: Based on the evaluation results of a medium-sized workflow, one can conclude that the cost results of Montage are much smaller than the cost results of Inspiral as in small-sized workflows. All the algorithms achieved less monetary cost compared to HEFT. When the workflow is Montage, the MCD achieves lower cost values than CFMAX and CCR. However, MCED cost results are lower than CFMAX and CCR only when the pricing model is Superlinear and sublinear. The same as in small-sized workflow, for both tested workflows, when the pricing model is super linear, there is no visible difference between the cost results generated by the competing algorithms for all the number of CR tested. It is also visible that when the number of CR=10, the algorithm generates fewer cost values in all cases and workflows. When the workflow is Inspiral, there is no visible difference between the cost results produced by our algorithms and those produced by CFMAX and CCR for the superlinear pricing model. In terms of makespan, we can conclude that the makespan results of the proposed algorithm start missing the deadline except when CR=5. It is worth mentioning that Inspiral requires high makespan than the one required by Montage.

3) Big-size workflow: Based on the evaluation results of a big-sized workflow, one can conclude that cost results of Montage are still much smaller than cost results of Inspiral as in small-sized and medium-sized workflows. All the algorithms achieved less monetary cost compared to HEFT as it is designed for makespan optimization. It is also still visible that when the number of *CR=5*, the algorithm generates fewer cost values in all cases and workflows. When the workflow is Montage, there is no visible difference between the cost results produced by our algorithms and the one produced by CFMAX and CCR for the super linear pricing model. The same results appear for Inspiral when the pricing model is sublinear for CR= [30 to 35], super linear for CR= [5 to 25]. In terms of makespan, one can conclude that the makespan results of the proposed algorithm misses the deadline except when CR= [5 to 10] for Montage, and when CR=5 for Inspiral. The distance between the deadline and the makespan is too big at this time because the workflow size is also big.

C. Run-time

We also studied the time overhead required by the proposed algorithm. Using an Intel(R) Core (TM) i3-2350M CPU @2.30GHz 2.30 GHz laptop and 4GB memory. We run the algorithm 100 times and collect the average time as the run-time result for both Montage and Inspiral. The run-time result for both Montage and Inspiral are shown in Fig. 3 and Fig. 4. It is easily visible that the run-time of the proposed algorithm is always low compared to the one of CFMAX and CCR.

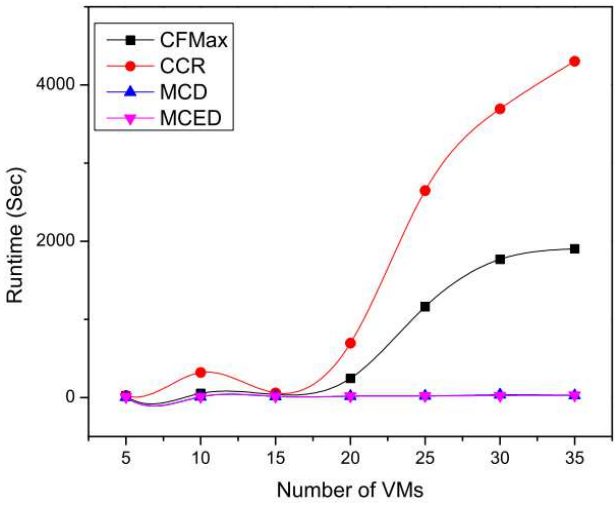

Fig. 3 Montage run-time: 100 tasks

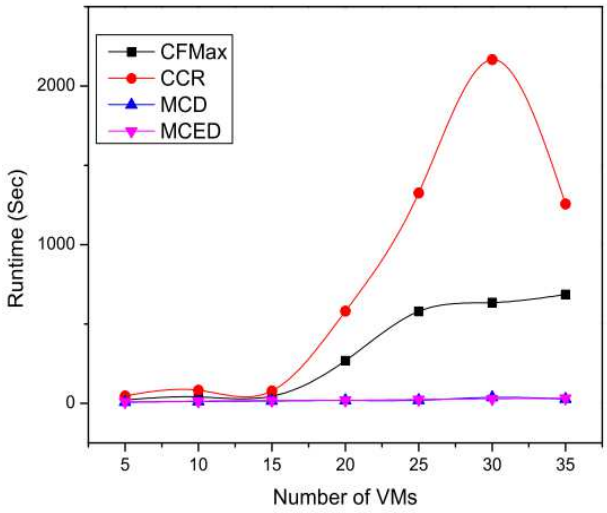

Fig. 4 Inspiral run-time: 100 tasks

Although our proposed algorithm is not constrained, all its makespan results are within the deadline for small-sized workflow applications. When the number of computing resources is small (cr=5) the proposed algorithm also performs well in terms of makespan and cost. The cost reduction rate of the proposed algorithm is not too high compared to the cost results of CFMAX and CCR, but with its low complexity and low run-time, this reduction rate is acceptable.

IV. Conclusion

In this paper, the problem of minimizing monetary cost for the execution of non-constrained workflows is considered, and an algorithm that selects a CR and tunes CPU frequency for each task is proposed to reduce the overall user cost. The evaluation results suggest that the proposed algorithm can significantly perform well for small-sized workflow and a small number of CR. Future works could consider the

inclusion of the deadline, energy and security will be some of the crucial problems to be taken care of. Their optimization may also be considered.

REFERENCES

[1] N. Mansouri, R. Ghafari, and B. M. H. Zade, "Cloud computing simulators: A comprehensive review," Simul. Model. Pract. Theory, vol. 104, p. 102144, 2020, doi: 10.1016/j.simpat.2020.102144.

[2] C. Tshimanga, K. Emmanuel, S. Ntumba, B. Eugène, and M. Mukendi, "A multi - criteria decision making heuristic for workflow scheduling in cloud computing environment," J. Supercomput., no. 0123456789, 2022, doi: 10.1007/s11227-022-04677-z.

[3] Y. Liu, A. Soroka, L. Han, J. Jian, and M. Tang, "Cloud-based big data analytics for customer insight-driven design innovation in SMEs," Int. J. Inf. Manage., vol. 51, no. November, pp. 0–1, 2020, doi: 10.1016/j.ijinfomgt.2019.11.002.

[4] Y. Gu and C. Budati, "Energy-aware workflow scheduling and optimization in clouds using bat algorithm," Futur. Gener. Comput. Syst., vol. 113, pp. 106–112, 2020, doi: 10.1016/j.future.2020.06.031.

[5] S. Azizi, M. Zandsalimi, and D. Li, "An energy-efficient algorithm for virtual machine placement optimization in cloud data centers," Cluster Comput., vol. 23, no. 4, pp. 3421–3434, 2020, doi: 10.1007/s10586-020-03096-0.

[6] L. Kong, J. Pepe, B. Mapetu, and Z. Chen, "Heuristic Load Balancing Based Zero Imbalance Mechanism in Cloud Computing," J. Grid Comput., vol. 18, no 1, pp. 123–148, 2019, [Online]. Available: doi.org/10.1007/s10723-019-09486-y

[7] A. Pujiyanta and L. Edi, "Job Scheduling Strategies in Grid Computing," Int. J. Adv. Sci. Eng. Inf. Technol., vol. 12, no. 3, pp. 1293–1300, 2022.

[8] P. Paknejad, R. Khorsand, and M. Ramezanpour, "Chaotic improved PICEA-g-based multi-objective optimization for workflow scheduling in cloud environment," Futur. Gener. Comput. Syst., vol. 117, pp. 12–28, 2021, doi: 10.1016/j.future.2020.11.002.

[9] "CloudSigma. Accessed on: , [Online]. Available: https://www.cloudsigma.com/us/.".

[10] H. R. Faragardi, M. R. Saleh Sedghpour, S. Fazliahmadi, T. Fahringer, and N. Rasouli, "GRP-HEFT: A Budget-Constrained Resource Provisioning Scheme for Workflow Scheduling in IaaS Clouds," IEEE Trans. Parallel Distrib. Syst., vol. 31, no. 6, pp. 1239–1254, 2020, doi: 10.1109/TPDS.2019.2961098.

[11] V. Kelefouras and K. Djemame, "Workflow simulation and multi-threading aware task scheduling for heterogeneous computing," J. Parallel Distrib. Comput., vol. 168, pp. 17–32, 2022, doi: 10.1016/j.jpdc.2022.05.011.

[12] S. Saeedi, R. Khorsand, S. Ghandi Bidgoli, and M. Ramezanpour, "Improved many-objective particle swarm optimization algorithm for scientific workflow scheduling in cloud computing," Comput. Ind. Eng., vol. 147, p. 106649, 2020, doi: 10.1016/j.cie.2020.106649.

[13] J. Zhou, T. Wang, P. Cong, P. Lu, T. Wei, and M. Chen, "Cost and makespan-aware workflow scheduling in hybrid clouds," J. Syst. Archit., vol. 100, 2019, doi: 10.1016/j.sysarc.2019.08.004.

[14] F. Jauro, H. Chiroma, A. Y. Gital, M. Almutairi, S. M. Abdulhamid, and J. H. Abawajy, "Deep learning architectures in emerging cloud computing architectures: Recent development, challenges and next research trend," Appl. Soft Comput. J., vol. 96, p. 106582, 2020, doi: 10.1016/j.asoc.2020.106582.

[15] B. Liang, X. Dong, Y. Wang, and X. Zhang, "A low-power task scheduling algorithm for heterogeneous cloud computing," J. Supercomput., vol. 76, no. 9, pp. 7290–7314, 2020, doi: 10.1007/s11227-020-03163-8.

[16] E. Bugingo, D. Zhang, Z. Chen, and W. Zheng, "Towards decomposition based multi-objective workflow scheduling for big data processing in clouds," Cluster Comput., vol. 24, no. 1, pp. 115–139, 2021, doi: 10.1007/s10586-020-03208-w.

[17] X. Guo, "Multi-objective task scheduling optimization in cloud computing based on fuzzy self-defense algorithm," Alexandria Eng. J., vol. 60, no. 6, pp. 5603–5609, 2021, doi: 10.1016/j.aej.2021.04.051.

[18] J. E. Ndamlabin Mboula, V. C. Kamla, and C. Tayou Djamegni, "Cost-time trade-off efficient workflow scheduling in cloud," Simul. Model. Pract. Theory, vol. 103, no. October 2019, p. 102107, 2020, doi: 10.1016/j.simpat.2020.102107.

[19] Y. Hao, J. Cao, Q. Wang, and J. Du, "Energy-aware scheduling in edge computing with a clustering method," Futur. Gener. Comput. Syst., vol. 117, pp. 259–272, 2021, doi: 10.1016/j.future.2020.11.029.

[20] A. Mohammadzadeh, M. Masdari, and F. S. Gharehchopogh, Energy and Cost-Aware Workflow Scheduling in Cloud Computing Data Centers Using a Multi-objective Optimization Algorithm, vol. 29, no. 3. Springer US, 2021. doi: 10.1007/s10922-021-09599-4.

[21] K. Mishra and S. K. Majhi, "A binary Bird Swarm Optimization based load balancing algorithm for cloud computing environment," Open Comput. Sci., vol. 11, no. 1, pp. 146–160, 2021, doi: 10.1515/comp-2020-0215.

[22] M. Sardaraz and M. Tahir, "A parallel multi-objective genetic algorithm for scheduling scientific workflows in cloud computing," Int. J. Distrib. Sens. Networks, vol. 16, no. 8, 2020, doi: 10.1177/1550147720949142.

[23] C. G. Ralha, A. H. D. Mendes, L. A. Laranjeira, A. P. F. Araújo, and A. C. M. A. Melo, "Multiagent system for dynamic resource provisioning in cloud computing platforms," Futur. Gener. Comput. Syst., vol. 94, pp. 80–96, 2019, doi: 10.1016/j.future.2018.09.050.

[24] A. Asghari and M. K. Sohrabi, "Combined use of coral reefs optimization and multi-agent deep Q-network for energy-aware resource provisioning in cloud data centers using DVFS technique," Cluster Comput., vol. 25, no. 1, pp. 119–140, 2022, doi: 10.1007/s10586-021-03368-3.

[25] P. S. Rawat, P. Dimri, P. Gupta, and G. P. Saroha, "Resource provisioning in scalable cloud using bio-inspired artificial neural network model," Appl. Soft Comput., vol. 99, p. 106876, 2021, doi: 10.1016/j.asoc.2020.106876.

[26] X. Zhou, G. Zhang, J. Sun, J. Zhou, T. Wei, and S. Hu, "Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based HEFT," Futur. Gener. Comput. Syst., vol. 93, pp. 278–289, 2019, doi: 10.1016/j.future.2018.10.046.

[27] T. A. L. Genez, I. Pietri, R. Sakellariou, L. F. Bittencourt, and E. R. M. Madeira, "A Particle Swarm Optimization Approach for Workflow Scheduling on Cloud Resources Priced by CPU Frequency," Proc. - 2015 IEEE/ACM 8th Int. Conf. Util. Cloud Comput. UCC 2015, pp. 237–241, 2015, doi: 10.1109/UCC.2015.40.

[28] W. Ahmad, B. Alam, S. Ahuja, and S. Malik, "A dynamic VM provisioning and de-provisioning based cost-efficient deadline-aware scheduling algorithm for Big Data workflow applications in a cloud environment," Cluster Comput., vol. 24, no. 1, pp. 249–278, 2021, doi: 10.1007/s10586-020-03100-7.

[29] N. Rizvi and D. Ramesh, "Fair budget constrained workflow scheduling approach for heterogeneous clouds," Cluster Comput., vol. 23, no. 4, pp. 3185–3201, 2020, doi: 10.1007/s10586-020-03079-1.

[30] T. A. L. Genez, L. F. Bittencourt, and E. R. M. Madeira, "Time-discretization for speeding-up scheduling of deadline-constrained workflows in clouds," Futur. Gener. Comput. Syst., vol. 107, pp. 1116–1129, 2020, doi: 10.1016/j.future.2017.07.061.

[31] N. Rizvi and D. Ramesh, "HBDCWS: heuristic-based budget and deadline constrained workflow scheduling approach for heterogeneous clouds," Soft Comput., vol. 24, no. 24, pp. 18971–18990, 2020, doi: 10.1007/s00500-020-05127-9.

[32] E. B. Edwin, P. Umamaheswari, and M. R. Thanka, "An efficient and improved multi-objective optimized replication management with dynamic and cost aware strategies in cloud computing data center," Cluster Comput., vol. 22, no. s5, pp. 11119–11128, 2019, doi: 10.1007/s10586-017-1313-6.

[33] K. Kalyan Chakravarthi, L. Shyamala, and V. Vaidehi, "Budget aware scheduling algorithm for workflow applications in IaaS clouds," Cluster Comput., vol. 23, no. 4, pp. 3405–3419, 2020, doi: 10.1007/s10586-020-03095-1.

[34] I. Pietri and R. Sakellariou, "Cost-efficient CPU provisioning for scientific workflows on clouds," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 9512, pp. 49–64, 2016, doi: 10.1007/978-3-319-43177-2_4.

[35] E. Saeedizade and M. Ashtiani, DDBWS: a dynamic deadline and budget-aware workflow scheduling algorithm in workflow-as-a-service environments, vol. 77, no. 12. Springer US, 2021. doi: 10.1007/s11227-021-03858-6.

[36] N. Zhou, W. Lin, W. Feng, F. Shi, and X. Pang, "Budget-deadline constrained approach for scientific workflows scheduling in a cloud environment," Cluster Comput., vol. 1, 2020, doi: 10.1007/s10586-020-03176-1.

[37] Y. Wen, J. Liu, W. Dou, X. Xu, B. Cao, and J. Chen, "Scheduling workflows with privacy protection constraints for big data applications on cloud," Futur. Gener. Comput. Syst., vol. 108, pp. 1084–1091, 2020, doi: 10.1016/j.future.2018.03.028.

[38] S. Yassa, R. Chelouah, H. Kadima, and B. Granado, "Multi-objective approach for energy-aware workflow scheduling in cloud computing

environments," Sci. World J., vol. 2013, 2013, doi: 10.1155/2013/350934.
[39] G. Khojasteh Toussi and M. Naghibzadeh, "A divide and conquer approach to deadline constrained cost-optimization workflow scheduling for the cloud," Cluster Comput., vol. 24, no. 3, pp. 1711–1733, 2021, doi: 10.1007/s10586-020-03223-x.
[40] Y. Pan et al., "A Novel Approach to Scheduling Workflows Upon Cloud Resources with Fluctuating Performance," Mob. Networks Appl., vol. 25, no. 2, pp. 690–700, 2020, doi: 10.1007/s11036-019-01450-0.
[41] R. Valarmathi and T. Sheela, "Ranging and tuning based particle swarm optimization with bat algorithm for task scheduling in cloud computing," Cluster Comput., vol. 22, no. s5, pp. 11975–11988, 2019, doi: 10.1007/s10586-017-1534-8.
[42] P. Lu, G. Zhang, Z. Zhu, X. Zhou, J. Sun, and J. Zhou, "A review of cost and makespan-Aware workflow scheduling in clouds," J. Circuits, Syst. Comput., vol. 28, no. 6, 2019, doi: 10.1142/S021812661930006X.
[43] W. Zheng, Y. Qin, E. Bugingo, D. Zhang, and J. Chen, "Cost optimization for deadline-aware scheduling of big-data processing jobs on clouds," Futur. Gener. Comput. Syst., vol. 82, pp. 244–255, 2018, doi: 10.1016/j.future.2017.12.004.
[44] H. Topcuoglu, S. Hariri, and M. Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," IEEE Trans. Parallel Distrib. Syst., vol. 13, no. 3, pp. 260–274, 2002, doi: 10.1109/71.993206.
[45] B. Emmanuel, Y. Qin, J. Wang, D. Zhang, and W. Zheng, "Cost optimization heuristics for deadline constrained workflow scheduling on clouds and their comparative evaluation," Concurr. Comput. Pract. Exp., vol. 30, no. 20, pp. 1–14, 2018, doi: 10.1002/cpe.4762.
[46] "Workflow Galler. Accessed on: April 20, 2021, [Online]. Available: https://confluence.pegasus.isi.edu/display/pegasus/." .