Applied Intelligence (2021) 51:1629–1644
https://doi.org/10.1007/s10489-020-01875-1

ORIGINAL SUBMISSION

Cost-effective workflow scheduling approach on cloud under deadline constraint using firefly algorithm

Koneti Kalyan Chakravarthi[1] · **L. Shyamala**[1] · **V. Vaidehi**[2]

Published online: 3 October 2020

Abstract

Cloud computing, a novel and promising methodology in the distributed computing domain, provides a pay-per-use framework to solve large-scale scientific and business workflow applications. These workflow applications have a constraint that each of them must completed within the limited time (deadline constraint). Therefore, scheduling a workflow with deadline constraints is increasingly becoming a crucial research issue. However, many analytical reviews on scheduling problems reveal that existing solutions fail to provide cost-effective solutions and they do not consider the parameters like CPU performance variation, delay in acquisition and termination of Virtual Machines (VMs). This paper presents a Cost-Effective Firefly based Algorithm (CEFA) to solve workflow scheduling problems that can occur in an Infrastructure as a Service (IaaS) platform. The proposed CEFA uses a novel method for problem encoding, population initialization and fitness evaluation with an objective to provide cost-effective and optimized workflow execution within the time limit. The performance of the proposed CEFA is compared with the state-of-the-art algorithms such as IaaS Cloud-Partial Critical Path (IC-PCP), Particle Swarm Optimization (PSO), Robustness-Cost-Time (RCT), Robustness-Time-Cost (RTC), and Regressive Whale Optimization (RWO). Our experimental results demonstrate that the proposed CEFA outperforms current state-of-the-art heuristics with the criteria of achieving the deadline constraint and minimizing the cost of execution.

Keywords Deadline constraint · Workflow scheduling · Scientific workflows · Firefly

1 Introduction

Complex scientific applications like Physics, Bioinformatics, Earth Science, Astronomy and disaster modeling can be represented naturally in the form of workflows [1, 2]. One of the benefits of workflow representation is that the workflow can be reusable, reproducible and even traceable through other workflows [3, 4]. Workflows, depicted as Directed Acyclic Graph (DAG), consists of computational activities interconnected through data-flow and control-flow dependencies [5]. Scientific workflows are partitioned into multiple tasks which require complex data of different sizes and tens of hundreds of processing hours [6]. In the meantime, computing systems where catastrophe can occur will become useless, if the completion of workflow execution takes more than some specified time. These workflow-based applications are highly demanding and challenging for processing huge amounts of data in real-time workflow tasks with the desired cost reduction of computing resources [7].

Cloud provides computing as a utility-based IT resource over the internet where users pay based on the actual consumption of computing resources [8]. A cloud service provider's basic offerings are categorized as a. Software as a Service, widely used for remote software application services (SaaS model) b. Platform as a Service that provides an application development platform for building and deploying the applications (PaaS model) and c. Infrastructure as a Service that provides middleware associated computing instances (IaaS model) [9]. SaaS and PaaS based solutions are not feasible for workflow applications since they only provide a platform to design, develop, and porting legacy applications to the new platform [10]. On the other hand, IaaS offers several benefits over traditional distributed environments [11]. *Direct on-demand* provisioning allows the users to directly provision/de-provision the required computing resources. *Elasticity*

✉ Koneti Kalyan Chakravarthi
kalyan.koneti@gmail.com

[1] School of Computer Science and Engineering, Vellore Institute of Technology, Chennai, India

[2] Mother Teresa Women's University, Kodaikanal, Tamil Nadu, India

offers the flexibility of procuring or releasing the computing resources with varying configurations. Therefore, for any scientific workflow application, the resource pool can grow or shrink based on its requirement.

In the cloud, the workflow is executed in two phases. In the first phase, resources are identified and acquired from the cloud to run the workflow tasks. The second phase generates a schedule by mapping each task to a suitable resource so that Quality of Service (QoS) requirements such as deadline, task precedence constraints are met. Previous works on the workflow scheduling in traditional distributed systems are mainly focused on the resource provisioning phase because these distributed environments provide a static pool of resources whose configuration is known in advance. Although, most of the researchers in traditional distributed systems focused on the minimization of makespan, while researchers in the cloud focus on other important parameters such as economic cost, consumption of energy and secure computation [12] besides execution time.

Existing works on the workflow scheduling problem assume that the monetary cost for a computation is based on the amount of actually used resources [13]. With this assumption, two key corollaries are 1) the total cost of a workflow is the sum of the costs of all sub-tasks, and 2) the cost of a task is fixed when running on certain service. On the other hand, in the cloud environments, the cost is determined by the running time of the underlying hosting instances. In addition, the runtime is usually measured by counting fixed-size time intervals, with the partially used intervals rounded up. Such schemes make the cost caused by a task hard to be precisely predicted before scheduling. For example, a task that shares the same time interval with the previous task hosted in the same instance might not produce extra cost. On the other hand, for a task which starts a new time interval but does not use it entirely, the cost might be more than the estimated.

In scheduling, there are still some specific challenges that need to be addressed. First, the CPU performance variation due to virtualization, shared and heterogeneous nature of non-virtualized hardware. Schad et al. [14] reported CPU performance variation up to 24% in Amazon's EC2 cloud. To make an allocation decision, many of the scheduling policies depend on the estimation of the task's runtime on various types of resources, assuming the capacity of the resource is always optimal. Due to the performance variation in the virtual machine, the actual completion of the task takes longer and is delayed. This leads to a deluge on the sub-tasks execution and may cause an application's deadline to be missed, or an increase in budget. So, the performance variation of the resources has a substantial impact on workflow scheduling. Secondly, acquiring a resource requires some time for proper initialization (acquisition delay) before it is available to the user. Likewise, when the resource is terminated, an approximate shutdown time is required (termination delay). Therefore, longer acquisition times may cause an application to miss its deadline or increase in budget. Third, there is a limitation in traditional heterogeneous environments with respect to the availability of the number of resources and their type. All the existing list-based heuristics allocate the best suitable resource to the task by traversing the entire list of available resources [13]. Since the cloud provides an infinite pool of resources with various configurations and pricing schemes, it is not possible to traverse an entire pool of resources. For example, PSO defines particle position and velocities as t x r matrices, where t is the number of tasks and r is the number of resources. However, r would be too large for the cloud. The typical encoding schemes usually represent the mapping of tasks to resources that might not be feasible for the cloud, since the resources acquired and released dynamically. These encoding schemes result in unnecessary high time and space complexity. The scheduling algorithm should take a suitable decision in choosing the resources in view of performance and cost optimization. All the above challenges of the cloud dictate the development of novel resource provisioning and workflow scheduling algorithms.

Workflow scheduling is widely known to be an NP-hard [17] problem. To solve the nature of NP-hard problem, heuristic algorithms are more suitable than the deterministic algorithms [15]. The parallel nature working of the Firefly Algorithm (FA) resolves the optimization problems. The 'lower level' (heuristic) in FA focuses on generating new solutions within the search space and thus selects the best solution for survival. On the other hand, Randomization, allows the search process to prevent the solution being trapped in local optima. The local search improves a candidate solution until improvements are detected, i.e., places the solution in local optimum.

In this paper, an attempt is made to use the Firefly algorithm to solve the workflow scheduling problem in the cloud which gives cost-effective realistic schedules. This paper proposes a novel scheduling algorithm CEFA which schedules workflow tasks to the low-cost virtual machine. Our proposal CEFA considers all the challenges of cloud, such as on-demand resource provisioning, elasticity, and pay-per-use price model and it also focus on issues such as CPU performance variation and their acquisition delay. Major contributions are made in the research of the workflow scheduling process as follows.

- Through modeling the IaaS cloud and workflow, we formulate the workflow scheduling problem in the IaaS cloud as an optimization one that attempts to minimize the makespan and execution cost of workflows simultaneously.

- In order to effectively solve the workflow scheduling problem in clouds, we propose a new deadline-constrained scheduling algorithm named Cost-Effective Firefly based Algorithm (CEFA), by designing Novel schemes for workflow encoding, initial population, and fitness evaluation.
- We conduct extensive experiments on synthetic data of real scientific workflows to verify the effectiveness and efficiency of the proposed CEFA algorithm. Experimental results show that CEFA can achieve better cost-makespan tradeoffs compared to three existing algorithms, including IC-PCP, PSO, RCT, RTC, and RWO.

The rest of the paper is organized as follows. Section 2 of this paper provides a precise study of literature on the work carried in the area of workflow scheduling and highlights the major works and proposals. Section 3 gives an overview the virtual machine modeling, workflow modeling and problem formulations, followed by the fundamentals of firefly algorithm in Section 4. In continuation, the proposed solution to the workflow scheduling problem shall be discussed in Section 5. Next is Section 6, which delivers the evaluation report of the arrived results and its comparison charts. Towards the end of this paper, in Section 7, the possible scope of future work in the workflow scheduling is suggested.

2 Related work

A multitude of studies have been made on scheduling problems in distributed systems and is NP-hard [17] in which a feasible schedule can't be found in polynomial time unless NP = P. Various heuristic and meta-heuristic approaches focused on generating the near-optimal schedules [18–23]. However, these solutions focused on meeting the QoS requirements while generating a schedule. This section summarizes the bibliographic review of several scheduling approaches in the IaaS cloud with the user's QoS requirements.

Workflow scheduling approaches are classified as either Heuristic or Meta-heuristic and then sub-categorized according to workflow multiplicity and scheduling techniques as shown in Fig. 1.

In heuristic-based workflow scheduling, Mao and Humphrey [24] proposed a Scaling-Consolidation-Scheduling (SCS) algorithm to execute workflow ensembles on the cloud. They acknowledge that there are various types of VMs with different prices and that they can be leased on demand, depending on the application's requirements. Furthermore, they tailor their approach so that the execution cost is minimized based on the cloud's pricing model, that is, VMs are paid by a fraction of time, which in most cases is one hour. They try to minimize the execution cost by applying a set of heuristics such as merging tasks into a single one, identifying the most cost-effective VM type for each task and consolidating instances. Although this is a valid approach capable of reducing the execution cost of workflows on clouds, the solution proposed only guarantees a reduction on the cost and not a near-optimal solution.

Another work on workflow ensembles developed for cloud is presented by Malawski et al. [25]. They proposed three algorithms in which two are dynamic algorithms, DPDS (Dynamic Provisioning Dynamic Scheduling), WA-DPDS (Workflow-Aware DPDS) and one static algorithm is SPSS (Static Provisioning Static Scheduling). These algorithms aim to maximize the number of workflows completed under QoS constraints such as deadline and budget. Their solutions acknowledge different delays present when dealing with VMs leased from IaaS cloud providers such as instance acquisition and termination delays. Furthermore, their approach is robust in the sense that the task's estimated execution time may vary based on

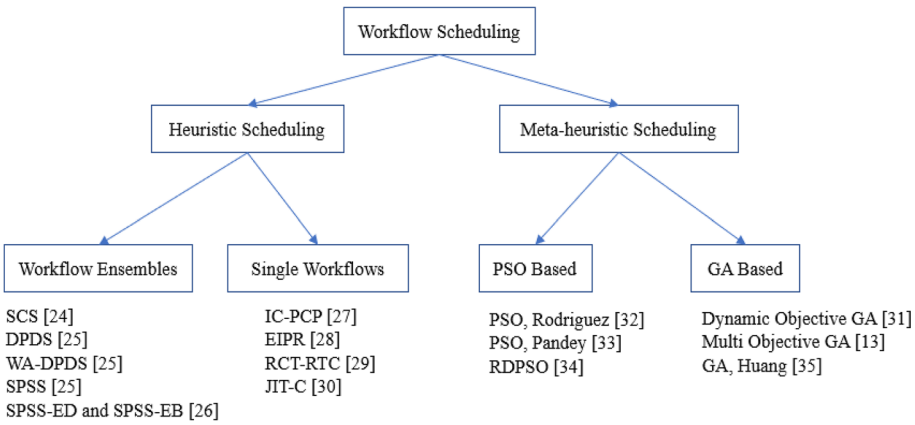


Fig. 1 Classification of scheduling algorithms

a uniform distribution and they use a cost safety margin to avoid generating a schedule that goes over budget. Their work, however, considers only a single type of VM, ignoring the heterogeneous nature of IaaS clouds.

Static Provisioning-Static Scheduling under Energy and Deadline Constraints (SPSS-ED) and Static Provisioning-Static Scheduling under Energy and Budget Constraints (SPSS-EB) are two energy-aware resource provisioning algorithms proposed by Pietri et al. [26] to schedule workflow ensembles based on Static Provisioning-Static Scheduling (SPSS). SPSS-ED focuses on meeting a deadline and energy constraints and SPSS-EB focuses on budget and energy constraints. Both of the algorithms aim to reduce the consumption of energy and increase the number of completed workflows. SPSS-ED plan it's execution by scheduling each task of the workflow. SPSS-ED plan is accepted only if the deadline and energy constraints are satisfied. In SPSS-EB, the same process is repeated but instead of the deadline, budget constraint is used. However, they do not account for CPU performance variations, different workflow structures, data transfer costs, and instance acquisition and termination delays.

Further, Heuristic based workflow scheduling algorithms are presented in [27–30] to execute a single workflow in the IaaS cloud. The work in [27–29] are based on the partial critical path (PCP) of the workflow. These approaches reduce the execution cost by scheduling all the workflow tasks in a critical path on a single machine. However, they do not have a global optimization strategy in place that can provide a near-optimal solution; instead, they use a task level optimization and hence fail to generate a better solution for the whole workflow.

Abrishami et al. [27] formulated the IaaS Cloud Partial Critical Path (IC-PCP) algorithm, which is a non-robust deadline constrained algorithm. This algorithm schedules a workflow in an IaaS offering whilst reducing costs involved in execution within the user-specified application's deadline. It follows Partial Critical Paths (PCPs) and hence, it starts identifying the critical paths which are associated with the exit node of the workflow. Then, it assigns each task of the critical path on to the cheapest resource, which can execute before its finish time. If in case, this cannot be achieved, the cheapest resource that can finish the tasks before their latest finish time is leased and PCP assigned to it. The procedure is executed recursively until all the tasks are scheduled. They try to minimize the execution cost based on the heuristic of scheduling all tasks in a partial critical path on a single machine which can finish the tasks before their latest finish time (which is calculated based on the application's deadline and the fastest available instance). However, they do not have a global optimization technique in place capable of producing a near-optimal solution; instead, they use a task level optimization and hence fail to utilize the whole workflow structure and characteristics to generate a better solution.

Poola et al. [29] proposed RCT and RTC algorithms, which are also based on Partial Critical Paths (PCPs). These algorithms are robust, fault-tolerant and handle the performance variation and failures of the cloud resources. These algorithms schedule tasks on two diverse cloud instances, on-demand and spot (dynamic) instances by adapting just-in-time heuristics in scheduling. On the other hand, to handle performance variation of the resources, some amount of relaxed time slots are identified depending on the type of robustness and are driven by PCP execution time and its levels of tolerance to fluctuations in PCP. Also, to enhance the tolerance level a checkpoint is imposed in a gap of intervals. When a task fails, the algorithm resumes the task from the last checkpoint. However, the algorithm handled uncertainty by assuming the known distribution function for all the input data; besides increasing the infrastructural costs as the storage of checkpoints becomes mandatory. The comparison of our proposed work has been done with the robust scheduling algorithm only for the parameters which caters for performance variation and the acquisition delays of VMs. Furthermore, we have selected RCT and RTC resource selection policies as our baseline algorithms.

Just-In-Time (JIT-C) workflow scheduling algorithm for the Cloud environment is presented by Sahni et al. [30], is a dynamic cost-effective scheduling approach. If the deadline factor is achievable, then the optimal solution is generated. Otherwise, the user has to prompt the deadline again. Their algorithm exploits the advantages of the cloud computing as well as the performance variation of the resources and the acquisition delay.

Meta-heuristic algorithms [13, 31–35] are the other category of workflow scheduling algorithms. Modern meta-heuristics are accurate and efficient to find a "good enough" solution in a "small enough" computing time. Population-based metaheuristics find good solutions by iteratively selecting and then combining existing solutions from a population set by mimicking the principles of natural evolution. Commonly used population-based algorithms are the Particle Swarm Optimization (PSO), Genetic Algorithm (GA) and the recently developed algorithm called the Firefly Algorithm (FA) shows a lot of potential [36, 37] in workflow scheduling.

The authors of [32–34] solved the workflow scheduling problem using Particle Swarm Optimization (PSO). Pandey et al. [33] explicated an approach that lowers the cost of execution with load balancing in available machines. The execution time of the workflow is not considered in the scheduling objectives and therefore this value can be considerably high as a result of the cost minimization policy. The authors do not consider the elasticity of the cloud and assume a fixed set of VMs is available beforehand. For this

reason, the solution presented is similar to those used for grids where the schedule generated is a mapping between tasks and resources instead of a more comprehensive schedule indicating the number and type of resources that need to be leased, when they should be acquired and released, and in which order the tasks should be executed on them.

Rodriguez and Buyya [32] proposed a meta-heuristic optimization based approach, Particle Swarm Optimization (PSO) that gives promising results, but there are still rooms for enhancement. In their approach, particles are encoded based on the resources index that depicts the position of the particles. Because the index does not have much information on resources, particles move in a different dimensions to individual best, and global best does not lead to an optimal solution. This algorithm focused on the essential cloud characteristics such as the elasticity, pay as you go pricing model and heterogeneous nature of the unlimited resources, including the disparities in performance and the acquisition and termination delay of VMs. In this work, the considered PSO parameters are: the acceleration coefficient (c_i=acceleration coefficient; i = 1, 2), number of particles (n), and inertia factor or weight (ω). In this computation, the acceleration coefficient (c_i) was varied from 1.5 to 2.0, and the inertia (inertia factor or weight) ranged from 0.1 to 1.0. The number of particles is set to 100.

Wu et al. [34] proposed Revised Discrete Particle Swarm Optimization (RDPSO) which focuses on minimizing either cost or time while meeting a deadline or budget constraints. But it assumes an initial set of resources available beforehand and hence lacks in utilizing the elasticity of IaaS clouds.

Reddy and Kumar [47] proposed the Regressive Whale Optimization (RWO) algorithm for workflow scheduling in the cloud computing environment. RWO is the meta-heuristic algorithm, which schedules the task depending on a fitness function. The fitness function is defined based on resource utilization, energy, and the Quality of Service (QoS) constraints. The proposed algorithm differs from the standard Whlae Optimization Algorithm (WOA) in the position update step, where the modification is made with the introduction of a regression-based position update. In their approach, the solution is generated at random without considering the resource type. Resource Type provides the necessary information about VM which in turn helps to comply with the deadline factor. Also, it ignores the essential cloud characteristics such as the elasticity, pay as you go pricing model, and heterogeneous nature of the unlimited resources, including the disparities in performance and the acquisition and termination delay of VMs.

Furthermore, approaches presented by Huang [35], Zhu et al. [13] and Chen et al. [31] are based on Genetic Algorithms. Chen et al. [31] proposed a cost optimization strategy under deadline constraint that uses the same encoding approaches as Rodriguez and Buyya [32]. When there is no optimal/feasible solution, their strategy moves from cost to time function to minimize the execution time, but not the cost. Zhu et al. [13] and Huang [35] focused on essential characteristics of the cloud by ignoring the CPU performance variation of the resources and the acquisition delay. However, these algorithms need to evolve for numerous generations and a feasible solution may not be found.

In the literature, numerous approaches are proposed on heuristic and meta-heuristic algorithms. However, meta-heuristic algorithms outperform other algorithms when the number of tasks is less and fall behind if the number of tasks is more. The meta-heuristic algorithms outlined do not conform to the basic principles of the cloud. So, this work proposes a Cost-Effective Firefly based Algorithm by addressing the challenges, such as the CPU performance variation of the resources, the acquisition delay and attempts to optimize the cost under deadline constraint.

3 Modeling and problem formulation

In this section, we first give the model of virtual machines (VMs), workflow and then formulate the scheduling problem.

3.1 Virtual machine modeling

The cloud platform provides an infinite number of VMs with various configurations. Let $VT = \{vt_1, vt_2, vt_3 \ldots, vt_m\}$ represent m types of virtual machines available in the cloud. The parameter $vm_k^{vt_u}$ represents the k^{th} VM of type vt_u. Also, each virtual machine VM $vm_k^{vt_u}$ is associated with an estimated lease begin time $LBT_{vm_k^{vt_u}}$ and lease end time $LET_{vm_k^{vt_u}}$. The price of the virtual machine denoted as $price(vm_k^{vt_u})$ is the cost per unit interval of time. It is worth noting that virtual machines can be acquired and terminated at any point in time. Also, virtual machines are charged per unit interval of time, and any partial use of the unit of time will be charged for the whole period.

3.2 Workflow modeling

In the cloud, a workflow (W) can be modeled as $W = \{G, w_d\}$, where G and w_d represent workflow's structure and deadline constraint respectively. The structure G of workflow W can be formally modelled as a directed acyclic graph (DAG), i.e., $G = (T, E)$, where $T = \{t_1, t_2, \ldots, t_n\}$ is a set of tasks and the parameter n is the task count; the vertex t_j denotes the j^{th} task in a workflow W. Also, $E \subseteq T \times T$ denotes the directed edges among tasks. An edge $e_{k,j} \in E$ of the form (t_k, t_j) exists if there is a precedence constraint between the tasks t_k and t_j, where t_k is an immediate predecessor of task t_j and the task t_j is an

immediate successor of task t_k. The $pred(t_j)$ denotes the set of tasks consisting of all t_j's immediate predecessors, and $succ(t_j)$ represents the set of tasks consisting of all t_j's immediate successors. A simple workflow is shown in Fig. 2

3.3 Problem formulation

In workflow scheduling, let $ET_{j,k}$ denotes the execution time of a task t_j on VM $vm_k^{vt_u}$. The execution time of a task t_j on various instances are calculated based on the task's size ($Size(t_j)$) divided by Processing Capacity ($PC_{vm_k^{vt_u}}$) of the instance type vt_u.

$$ET_{j,k} = \frac{Size(t_j)}{PC_{vm_k^{vt_u}}} \tag{1}$$

Additionally, CPU performance variation of the resource is modeled by altering the processing capacity of the VM $vm_k^{vt_u}$ by introducing a performance degradation ($PerDeg_{vm_k^{vt_u}}$) parameter. So, (1) is updated as

$$ET_{j,k} = \frac{Size(t_j)}{(PC_{vm_k^{vt_u}} * (1 - PerDeg_{vm_k^{vt_u}}))} \tag{2}$$

Further, data transfer time $DT_{j,k}$ across the tasks on different virtual machines is the ratio of output-data file size (d_{t_j}) to average bandwidth capacity β within the same data center in the cloud, as shown in (3). The data transfer time $DT_{j,k}$ becomes zero when both parent and child tasks are scheduled on the same VM.

$$DT_{j,k} = \frac{d_{t_j}}{\beta} \tag{3}$$

Finally, total processing time $PT_{j,k}$ of task t_j on $vm_k^{vt_u}$ is obtained using (4). Where e is the out degree of the task t_j and $IsVM_{Same}$ is zero whenever t_j and t_k run on the same VM or 1 otherwise.

$$PT_{j,k} = ET_{j,k} + \sum_{1}^{e} \left(DT_{e_{jk}} * IsVM_{Same}\right) \tag{4}$$

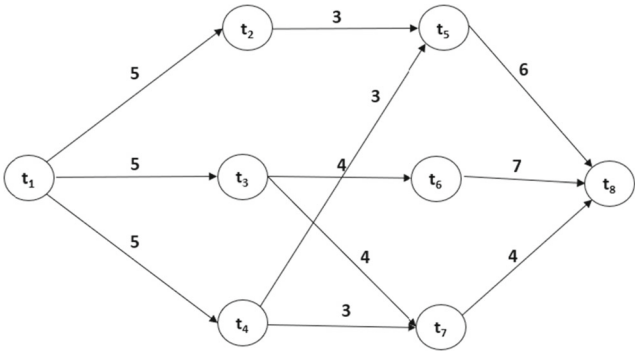


Fig. 2 An example workflow

Besides, the symbols $EST_{j,k}$ and $EFT_{j,k}$ denotes the earliest start time and earliest finish time of a task t_j on VM $vm_k^{vt_u}$. The earliest time at which a task t_j can begin its execution on VM $vm_k^{vt_u}$ is known as the earliest start time and calculated as follows.

$$EST_j = \begin{cases} 0, & \text{if pred}(t_j) = NULL \\ \max\limits_{t_p \in \text{pred}(t_j)} \left\{EST_p + MET_p + MTT_{p,j}\right\}, & \text{otherwise} \end{cases} \tag{5}$$

Where $MTT_{p,j}$ denotes the minimum data communication time from predecessor task t_p to current task t_j and MET_j represent the minimum execution time of a task t_j on VM $vm_k^{vt_u} \in VT$ that have minimum execution time among all VM types in the cloud and computed as

$$MET_j = \min_{vm_k^{vt_u} \epsilon VT} \left\{ET_{j,k}\right\} \tag{6}$$

Evidently, the estimated finish time, EFT_j can be computed as

$$EFT_j = EST_j + MET_j \tag{7}$$

In an arbitrary workflow scheduling, the latest finish time LFT_j of task t_j is the period before which task completes its computation, such that finish time of workflow W is less than the user-specified deadline, w_d. It is defined as

$$LFT_j = \begin{cases} w_d, & \text{if succ}(t_j) = NULL \\ \min\limits_{t_p \in \text{succ}(t_j)} \left\{LFT_p - MET_p - MTT_{p,j}\right\}, & \text{otherwise} \end{cases} \tag{8}$$

Due to precedence constraints in a workflow, the task can't be executed until it gathers all of the data from its immediate predecessors.

$$FT_{p,k} + DT_{p,j} \leq ST_{j,k} \quad \forall e_{p,j} \in E \tag{9}$$

Where $FT_{p,k}$ denotes task t_p's finish time on VM $vm_k^{vt_u}$, $DT_{p,j}$ denotes the data transfer time between task t_p and t_j and $ST_{j,k}$ represent the start time of task t_j on VM $vm_k^{vt_u}$.

In workflow scheduling environments, finish time of workflow W_{FT} is the maximal finish time of all its tasks and is defined as

$$W_{FT} = \max_{t_j \in (W)} \left\{FT_{j,k}\right\} \tag{10}$$

The cost $c_{j,k}$ for executing task t_j on VM $vm_k^{vt_u}$ is calculated as

$$c_{j,k} = Price\left(vm_k^{vt_u}\right) * \left\lceil \frac{ET_{j,k}}{N_t} \right\rceil \tag{11}$$

Where $Price(vm_k^{vt_u})$ is the cost of the VM $vm_k^{vt_u}$ per one interval of time and N_t is the unit of time interval.

To ensure the deadline of a workflow, all the workflow tasks in W must finish their execution before its deadline

w_d. Consequently, this brings with it another constraint

$$W_{FT} \leq w_d \tag{12}$$

Subject to aforementioned constraints in (9) and (12), the primary goal of optimization is to minimize the Total Execution Cost (TEC) of completing workflow W, which can be computed as

$$\text{Minimize } TEC = \sum_{k=1}^{|VM|} Price(vm_k^{vt_u}) \cdot p_k \tag{13}$$

where |VM| represents the number of acquired VMs and p_k denotes the number of time units of leased VM $vm_k^{vt_u}$.

4 Firefly algorithm

Swarm based algorithms have received more attention in the research community due to advances in computing infrastructures. Fireflies are tiny winged beetles that belong to the family of Lampyridae. They are capable of flashing light to fascinate mates. They work like a capacitor, gradually charge to a definite threshold and discharge accumulated energy in the form of light. Based on this phenomenon Yang [37] developed the firefly algorithm.

The firefly algorithm has many advantages such as:

- ✓ The rate of convergence is maximum or maximizing.
- ✓ It can be used as a general as well as a global problem solver.
- ✓ Each firefly toils independently and settles with a superior position than its current position and the position of other fireflies. Hence, it is easy to escape from the local optima and scale to settle with global optimum.
- ✓ It is accomplished with the provision of diversified levels of robustness when compared to other meta-heuristic algorithms.
- ✓ It is simple, flexible, versatile and effectively efficient in addressing extensive range, but assorted real-time problems.

4.1 Inspiration

Firefly Algorithm (FA) developed by Yang [37, 46] represents one of the newest meta-heuristic techniques that idealizes the characteristics and behavior of the fireflies. The Firefly algorithm depends on three idealized principles [45]:

1. Fireflies have a unisex character; each firefly is fascinated by another, irrespective of gender.
2. Attractiveness factor is directly proportional to its brightness. A firefly with lesser brightness gets attracted to a firefly with more brightness.
3. The brightness of the firefly is represented by a fitness function that is maximized. This determines the quality of the workflow schedule.

In accordance with the previous rules, the basic Firefly algorithm is given in Algorithm 1.

Algorithm 1 Firefly.

```
Given :
Initialize the population of n fireflies
Objective/Fitness function f(y) where y = (y1, y2, y3,
... yd) where d represents dimension
Compute Intensity of Light Ii at yi using f (yi)
Output: Best Solution
while stopping requirements are not met do
    for j ← 1 to n do
        for k ← 1 to n do
            if (f(yj) > f (yk) then
                Move firefly j towards k using (16)
            end
        end
    end
    Update Light Intensity for all fireflies and Evaluate
    the new solution;
    Rank all Fireflies and find evaluate the best firefly
end
```

4.2 Firefly evaluation

Light intensity computation of firefly, highly relies on the nature of investigating the problem. In this scenario, the quality of the workflow schedule solution is computed by Total-Execution-Cost (TEC) using (13).

4.3 Distance

Let the distance between any two fireflies, firefly "i at y_i" and "j at y_j" be 'r_{ij}'. In the Cartesian framework, 'r_{ij}' is defined using (14). Where $y_{j,k}$ is the k^{th} component of spatial coordinate y_j associated with j^{th} firefly and d is the dimension of the investigating issue [46].

$$r_{ij} = \|y_i - y_j\| = \sqrt{\sum_{k=1}^{d} \left(y_{i,k} - y_{j,k}\right)^2} \tag{14}$$

4.4 Attractiveness

The attractiveness (β) of two fireflies depends on the distance (r) between fireflies and absorption coefficient of light (γ). The degree of attraction $\beta(r)$ of a firefly is estimated by (15).

$$\beta(r) = \beta_0 e^{-\gamma r^2} \tag{15}$$

4.5 Movement

The movement of the firefly k at y_k towards more attractive firefly j is established using (16).

$$y_{k+1} = y_k + \beta_0 e^{-\gamma r_{kj}^2}(y_k - y_j) + \alpha(rand - 0.5) \quad (16)$$

4.6 Efficacy of firefly algorithm

By inheritance, FA follows all the advantages that other swarm intelligence-based algorithms have. Upon analysis, a few insights that draw attention are as follows:

- ✓ FA has the potential to automatically divide the whole population into subgroups. This is because, invariably, local attraction strength is superior to long-distance attractions; resulting FA efficiently handles the extremely non-linear and multi-facet optimization problems naturally.
- ✓ FA believes neither the historical individual best nor the explicit global best. This draws the strength to resist the probable weakness of impulsive convergence. Additionally, FA does not rely on the velocities similar to that of velocities in PSO.
- ✓ FA is equipped to control and acclimatize to crisis settings by imposing control over the scaling constraint like, γ .

5 Proposed firefly based workflow scheduling

In the cloud computing, an efficient encoding mechanism can provide an efficient solution that fulfills the QoS constraints and raise efficiency. There are two steps involved in modeling the Firefly problem. First, the encoding mechanism used in defining the problem, that is, how initial solution is represented. Second, the fitness function that measures the quality of the solution.

5.1 Map between FA and workflow scheduling

A map between workflow scheduling and FA is provided before describing the proposed scheduling method. The proposed mapping would be as follows.

a. The dimension of the problem (d) is mapped to the total number of tasks in the workflow.
b. The location (y_i) of each firefly is mapped to a potential solution to the workflow scheduling problem.
c. Intensity (I) is mapped to the fitness of each solution of the workflow scheduling. Schedules with lower-cost are equivalent to fireflies of higher intensity.
d. Movement of low-intensity fireflies towards high-intensity fireflies is mapped to changing non-optimal schedules to more optimal schedules.

5.2 Firefly modelling

For the workflow scheduling scenario, the firefly represents workflow tasks; hence, the total number of tasks in workflow determines the size of a typical firefly. Workflow tasks execution order is determined based on dependency constraints of workflow and the index is assigned to each workflow task. A workflow schedule consists of 3 elements, first is task execution order (TaskPriorityIndex), second is allocation of the appropriate resource (TaskToResource) and third determines the resource type (ResourceType). Figure 3 shows the sample firefly encoding for the workflow in Fig. 2. This figure clearly shows the mapping of the task to a resource. For example, task t_3 will be executed at the virtual machine number 2 of type 1.

The fitness function should reflect the objectives of workflow scheduling, as it determines how "good" the potential solution is. The fitness function value is the overall execution cost for each of the associated schedule derived from the firefly's position.

5.3 Schedule generation

In order to handle the workflow scheduling problem, evaluation of total execution time and total execution cost of the considered workflow should be carried with an explicit task-resource mapping. The evaluation of the Total-Execution-Cost (TEC) and the Total-Execution-Time (TET) in a generic workflow is presented in Algorithm 2. Initialize an array VM_S for maintaining the provisioned resources. Now, this algorithm evaluates the execution time of a task $t_i \in T$ in each type of $vt_i \in VT$ using (2) and data transfer time $DT_{j,k}$, to transfer data from task t_j to t_k using (3). The Start Time of task t_j (ST_j) has two scenarios. If the task does not have any parents, then it can start its execution as soon as the virtual machine is allotted. Otherwise, the task starts as soon as the parents finish their execution and transfer their output. The finish time of the task t_j is obtained by adding ST_j to PT_j . Then update the resource in VM_S. This process continued until each workflow task is scheduled.

Fig. 3 Firefly Encoding for Sample Workflow (For Fig. 2)

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
TaskPriorityIndex	1	2	3	4	5	6	7	8
TaskToResource	1	1	2	3	1	2	3	2
ResourceType	1	2	1	3	3	2	1	2

5.4 Initial population

The execution time of tasks in the critical path affects the total execution time significantly, but the execution cost is a minor part of the total cost of execution. Therefore, assigning critical path tasks to faster virtual machines significantly reduces the total execution time, while the cost of execution has a limited impact on the total cost of execution. The diversity and veracity of the initial population of a firefly influence the performance greatly. To improve the convergence speed and solution quality, 20% of the initial population in the critical path [44] is assigned to the fastest virtual machines and the next 20% of the initial population is assigned to virtual machines with low processing capability. The rest of the population is generated randomly.

Algorithm 2 Schedule generation.

Input :
Tasks of Workflow T, Set of virtual machines (VT), An Array F[pos] = (FF^1_{pos}, FF^2_{pos} ,... FF^d_{pos}) representing Firefly's position
Output:
Schedule, TEC, TET
Initialize the matrix, VM_S represents the acquired resources
Compute Execution Time ET [| T| X | VT |] using (2)
Compute Data Transfer Time DT [|T|X|T|] using (3)
for $i \leftarrow 1$ **to** |$TaskPriorityIndex$| **do**
  $t_i = T[i]$
  $VM\,[i] = VT\,[\,pos^i\,]$
  if t_i *has no parents* **then**
    $ST_{t_i} = LET_{VM[i]}$
  else
    ST_{t_i} = max (max { $ET_{ta} + DT_{ta,i}$: ta ∈ parents (t_i)} , $LET_{VM[i]}$) ;
  end
  foreach *child* $t_c \in t_i$ **do**
    if "*child*" t_c *is assigned to different* $VM[i]$ **then**
      $TransferableTime$ += $DT\,[i, c]$
    end
  end
  $PT_{t_i} = ET\left[i, pos^i\right] + TransferableTime$
  $FT_{t_i} = PT_{t_i} + ST_{t_i}$
  if $VM\,[i]$ *not in* VM_S **then**
    $LBT_{VM[i]} = \max\left(ST_{t_i}, bootTime\right)$
    $VM_s = VM_s \cup VM[i]$
  end
  $LET_{VM[i]} = LBT_{VM[i]} + PT_{t_i}$
end
Compute TEC using (13)
Compute TET using (10)

6 Performance evaluation

This section presents the experiments conducted to evaluate the performance of the proposed approach.

6.1 Experimental workflows

The performance of CEFA is evaluated with different workflows used in different scientific fields: LIGO, Epigenomics, CyberShake, and Montage. These workflows have diverse structural properties such as pipeline, aggregation, distribution, and redistributions as well as different composition as shown in Fig. 4. Montage, an astronomy application stitches a series of images to create personalized sky mosaics. Montage tasks require high intense I/O and CPU with low processing capacity. The LIGO workflow aims to detect gravitational waves. This workflow requires a large memory with a high CPU. The Epigenomics is used in bioinformatics to automate genome sequencing operation. Moreover, these tasks demand high power computational processors with limited I/O regulations. Cybershake is best suited for simulating the earthquake hazards using synthetic seismograms. These workflow tasks require large memory and high CPU. To ease the evaluation of scheduling algorithms, Bharathi [43] developed a set of synthetic workflows of various sizes that resembles concrete scientific workflows. Synthetic workflows are characterized by DAG in XML format and are available in [42]. For assessing the results of the proposed algorithm in terms of performance, experiments are carefully designed and carried out for the above-specified workflows with a varying number of tasks: small (about 25 tasks), average (about 100 tasks) and large (about 1000 tasks).

6.2 Experimental settings

The cloud service providers provide various types of VMs with varying configurations. The VM configurations of EC2 cloud offerings [41] are shown in Table 1. It is assumed that for each type of VM, the processing capacity in terms of floating-point operations per second (FLOPS) is available from the provider or can be estimated [40]. The estimated execution time of workflow tasks in various types of virtual machines is obtained based on their processing capacity. The change in CPU performance of each VM is modeled based on the results presented by Schad et al. [14]. Also, each virtual machine's performance is reduced by a maximum of 24% based on the normal distribution. Its average mean is found to be 12% along with a 10% standard deviation. In a similar way, the data transfer time in the same data center is increased by a maximum of 19% [14], based on the normal distribution. Its average mean is

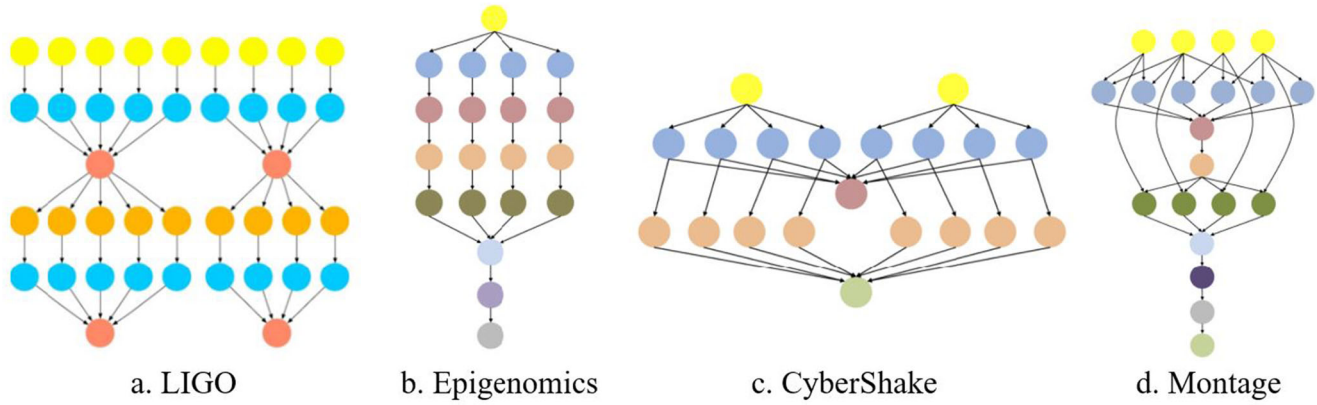


Fig. 4 Workflow structures differed in terms of characterization

found to be 9.5% along with a 5% standard deviation. The average bandwidth is set based on Amazon's Elastic Block Store [39] i.e., 20 MBps. The VM billing time is set to 10-minute interval and the estimated acquisition delay is set to one minute similar to Meena et al. [10].

In this work, the FA parameters considered are: the number of generations (G), number of fireflies (n), the coefficient of light absorption (γ), random variable (α) and initial attractiveness (β_0). These are selected in a random fashion and the combination factor ($n * G$) determines the extent of search in the solution space for obtaining the optimal solution. If the value of ($n * G$) is high, then it requires longer computational time, but it helps to get the best solution. In this computation, the coefficients of light absorption (γ) vary from '0' to '10' similar to Fister et al. [16], and the randomized parameter can have the fractional values from '0' to '1'.

To evaluate the performance metrics, two types of deadline constraints SoftDeadline and HardDeadline are considered. We introduce a deadline factor δ similar to Abrishami et al. [27] and vary from 0 to 3.2 with a step length of 0.4. For this, all workflow tasks are executed on the fastest resources and the minimum time to run the workflow W_{MET} is obtained. W_{MET} is the lower limit of the workflow execution time. The deadlines are established according to the rule specified in (17).

$$w_d = W_{MET} * (1 + \delta) \tag{17}$$

Table 1 VM Instance specifications

Type of VM	ECU	Memory(GiB)	Cost($/h)
m3.medium	1	3.75	0.067
c3.xlarge	4	3.75	0.21
m3.xlarge	4	15	0.266
c3.2xlarge	8	15	0.42
m3.2xlarge	16	30	0.532

Where W_{MET} is the minimum time required to execute the submitted workflow and δ is the deadline factor defined as follows.

For '*HardDeadline*': $0 \leq \delta \leq 1.2$
For '*SoftDeadline*': $1.2 \leq \delta \leq 3.2$
The deadline factor δ was varied with 0.4 disparity

6.3 Results analysis

The performance of the proposed CEFA is analyzed in terms of the compliance bounded by the deadline factor. The success rate for different scientific workflows under different deadlines is shown in Table 2. Table 2 presents the percentage of an improvement for experimental workflows with hard and soft deadline factors. Recently published algorithms IC-PCP [27], PSO [32], Robustness-Cost-Time [29], Robustness-Time-Cost [29], and RWO [47] are used as the baseline algorithms to evaluate the proposed solution.

6.3.1 Evaluation of deadline constraints

It can be seen from Table 2 that the proposed CEFA gives an improvement of 90.5%, 84.5%, 87.5% and 87.5% in Montage, Epigenomics, LIGO, and CyberShake respectively for hard deadline condition. It is also seen that for soft deadline conditions, the proposed CEFA gives 73%, 62%, 70.5% and 66.5% improvement for Montage, Epigenomics, LIGO, and CyberShake respectively.

From above, it is observed that IC-PCP has low performance in all reference algorithms for 'HardDeadline' and 'SoftDeadline' constraints. IC-PCP doesn't consider the performance degradation and delay in the acquisition of resources, which has a substantial effect on the execution cost and execution time, whereas the proposed CEFA considers performance degradation (PerDeg) and cloud resource startup time, which helps us to calculate the beginning and end time of task to ensure every task is executed under deadline constraint. The RTC and RCT

Table 2 Percentage of improvement for experimental workflows with deadline factor

Deadline factor	Algorithm	Montage		Epigenomics		LIGO		CyberShake	
		Success rate	Improvement % of CEFA	Success rate	Improvement % of CEFA	Success rate	Improvement % of CEFA	Success rate	Improvement % of CEFA
Hard deadline	IC-PCP	0	90.5	0	84.5	0	87.5	0	87.5
	RCT	47	43.5	37.5	47	53.5	34	40	43.5
	RTC	81.5	9	72.5	12	80	7.5	79.5	8
	RWO	85.5	5	78.5	6	83.5	4	82	5.5
	PSO	88.5	2	80	4.5	84	3.5	84.5	3
	CEFA	90.5	-	84.5	-	87.5	-	87.5	-
Soft deadline	IC-PCP	27	73	38	62	29.5	70.5	33.5	66.5
	RCT	52	48	53.5	46.5	56.5	43.5	49	51
	RTC	100	0	100	0	100	0	100	0
	RWO	100	0	100	0	100	0	100	0
	PSO	100	0	100	0	100	0	100	0
	CEFA	100	-	100	-	100	-	100	-

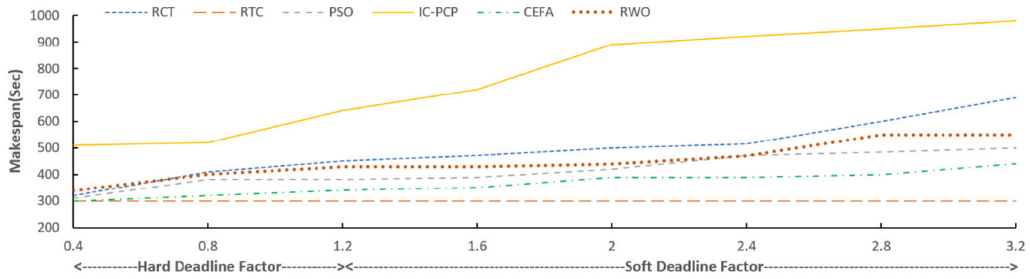
algorithms work on the policy of resource selection and their priorities are specified based on the Partial Critical Path (PCP) heuristics. Both can tolerate the CPU performance variation of the instance only to a certain extent.

In PSO, particles are encoded based on the resources index that depicts the position of the particles. Because the index does not have much information on the resources, particles tend to move in various directions towards the individual best, and global best may not lead to an ideal solution. In RWO, solutions are generated randomly based on the static resource pool and do not have much information on the resource types. RWO also ignores the CPU performance variation and the acquisition delay of the resources, which has a significant impact on the execution cost and execution time However, if the user specifies a hard deadline, then it's difficult to generate a feasible schedule for these approaches. CEFA encodes the firefly based on TaskPriorityIndex, TaskToResource mapping, and ResourceType. In TaskPriorityIndex, each task is assigned to an integer index based on its execution order. In TaskToResource, each task is mapped to a suitable resource and in ResourceType, vt_u type resource is selected from a pool of resources. This mechanism provides necessary information about the VM which in turn helps to comply with the deadline factor. Therefore, CEFA exceeds all baseline algorithms in respect of meeting the user-specified deadline factor.

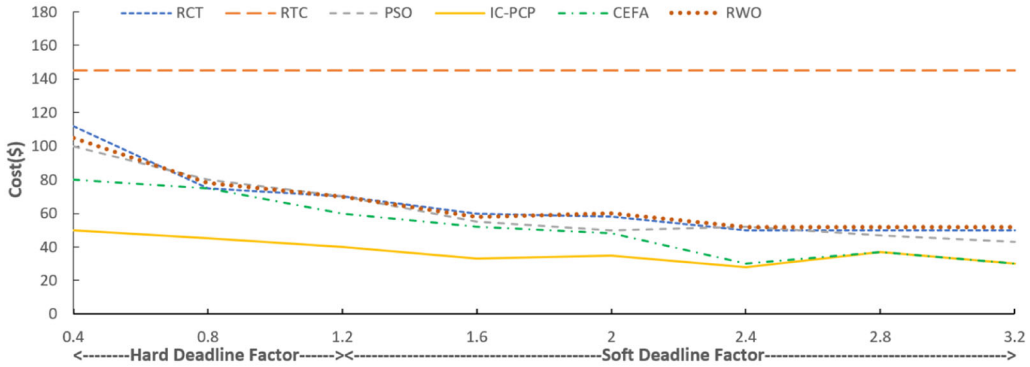
6.3.2 Makespan and cost evaluation (TET and TEC)

The proposed solution is anticipated to generate a schedule map which is cost-effective under deadline factor, therefore a comparison with the average of the makespan and the average cost should be observed. For each sample workflow, an average cost of execution (in $, in Fig. 5b, d, f, h) and total average execution time (in seconds) are depicted in Fig. 5a, c, e, g. The X-axis denotes the user-defined deadline factor values. If the deadline factor is between 0 and 1.2, then it is known as the hard deadline. Any value above 1.2 is known as a soft deadline.

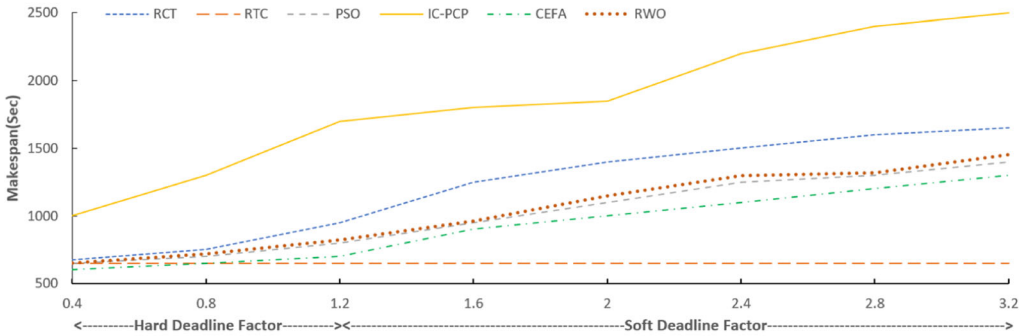
It can be seen in Fig. 5a, c, e, g, IC-PCP generates the cheapest schedule with the longest running time for each type of workflow deadline and hence, it does not meet any of the deadlines. On the other hand, the algorithm's goal is to devise an optimal and also a feasible schedule by minimizing the cost under the deadline factor. Therefore, an inexpensive schedule with a deadline factor violation is not useful. Therefore, the comparison is made among baseline algorithms that have the objective of meeting the deadlines. As seen in Fig. 5a, c, e, g, among PSO, RTC, RCT, RWO, and CEFA, the proposed algorithm generates the profitable schedule for each type of the workflow with an average success rate of 87.5% for hard deadline and 100% for a soft deadline.



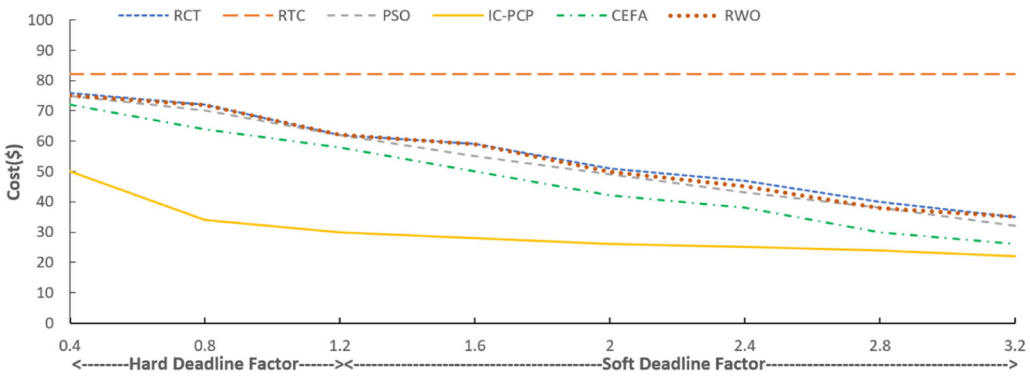
(a). Makespan for Montage Workflow



(b). Execution cost for Montage workflow

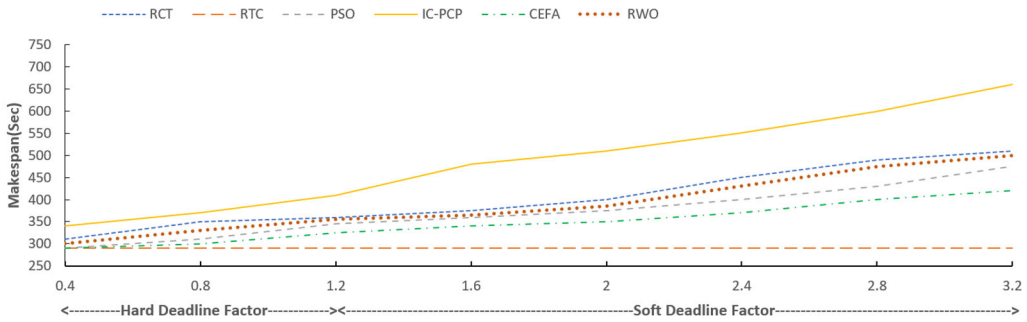


(c). Makespan for LIGO workflow

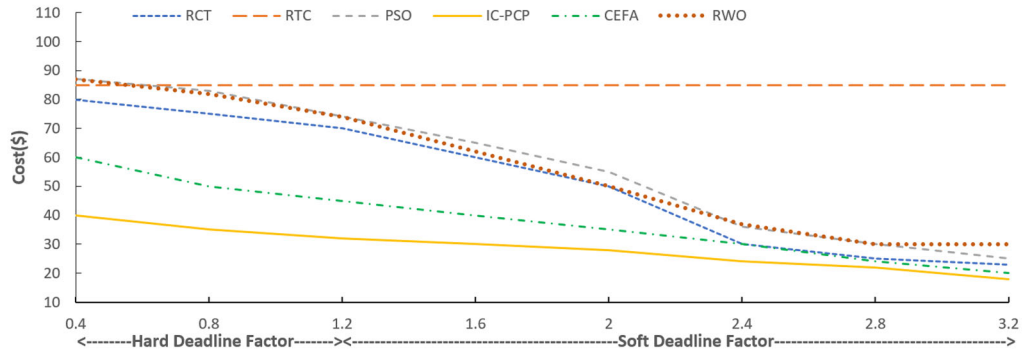


(d). Execution cost for LIGO workflow

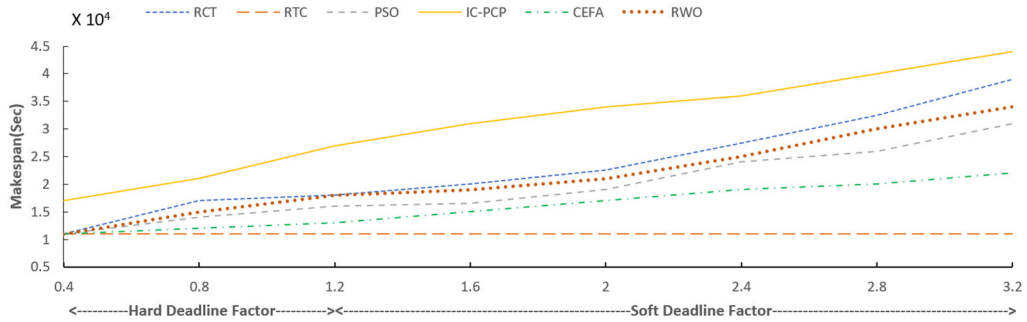
Fig. 5 **a**. Makespan for montage workflow. **b**. Execution cost for Montage workflow. **c**. Makespan for LIGO workflow. **d**. Execution cost for LIGO workflow. **e**. Makespan for CyberShake workflow. **f**. Execution cost for CyberShake workflow. **g**. Makespan for Epigenomics workflow. **h**. Execution cost for Epigenomics workflow



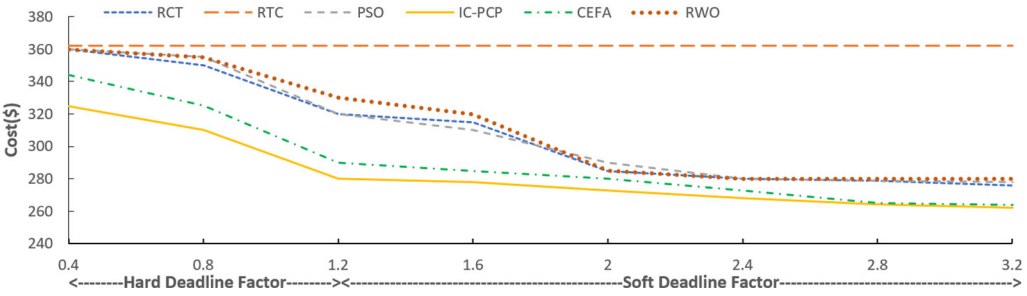
(e). Makespan for CyberShake workflow



(f). Execution cost for CyberShake workflow



(g). Makespan for Epigenomics workflow



(h). Execution cost for Epigenomics workflow

Fig. 5 (continued)

It can be seen from Fig. 5a for Montage workflow under deadline factor, CEFA gives an average lower makespan of 28%, 11%, 22%, 13% for RCT, PSO, RTC and RWO respectively. In a similar way, a lower makespan trend (Fig. 5c, e, g) is observed in LIGO, CyberShake, and Epigenomics workflows.

Figure 5b, d, f, h presents the cost incurred in the execution of Montage, LIGO, CyberShake, and Epigenomics for RTC,

RCT, PSO, RWO, and CEFA. It can be observed that the proposed CEFA cost is 30%, 21%, 24% lower than RCT, PSO, RWO respectively for Montage work flow. Similarly, a cost reduction trend is observed in LIGO, CyberShake, and Epigenomics.

Thus, results conclude that CEFA delivers better performance in comparison with baseline algorithms. For hard deadlines, CEFA is able to deliver the maximum percentage of success rates for all experimental workflows at a lower cost. However, as deadlines get relaxed, CEFA decreases the execution time and cost by capitalizing on the increased slack time.

6.3.3 Computational complexity

For deriving the computational complexity, it is assumed that scheduled workflow W is composed of k tasks and e edges. Since scheduled workflow W is represented as DAG, the total number of edges in the given workflow W is $k(k-1)/2 \approx \mathrm{O}\left(k^2\right)$. For each firefly generation, the distance, attractiveness and objective function of fireflies are evaluated. The extent of search in the solution space is determined by the overall available number of fireflies (N), along with their dimensions (d). The complexity of the objective function is determined by the schedule map algorithm and count of workflow tasks (T) and resources (R). As per the formulation d=k, the computational complexity of the recommending mechanism is $O\left(N * T^2 * R\right)$ per iteration. RCT, RTC, and IC-PCP are based on a heuristic approach. They execute much quicker than the proposed approach. While RCT, RTC, and IC-PCP have a polynomial time complexity, the time complexity of proposed CEFA is high which is similar to other meta-heuristic algorithms [38]. Though the proposed algorithm has got higher time complexity, it provides better schedules than the existing RCT, RTC, PSO,RWO, and IC-PCP algorithms. Hence this drawback can be tolerated.

7 Conclusion and future work

This paper presented a meta-heuristic Cost-Effective Firefly based Algorithm (CEFA) which minimizes the cost of execution under deadline constraint. Also, CEFA considers the parameters like CPU performance variation, delay in acquisition and termination of Virtual Machines (VMs) to attain the proposed objectives. For simulation, the CloudSim tool is used and the obtained results are compared with baseline algorithms such as IC-PCP, PSO, RWO, Robustness-Cost-Time, and Robustness-Time-Cost on diverse real-world scientific workflows. Observations reveal that the proposed scheme provides better results in terms of cost-effective realistic schedules. The proposed work can be extended in the real cloud environment. The present scheme considers the pricing from a single cloud service provider. This work can be extended for multiple pricing schemes from various cloud service providers.

References

1. Arabnejad V, Bubendorfer K, Ng B (2019) Budget and deadline aware e-science workflow scheduling in clouds. IEEE Trans Parallel and Distrib Sys 30(1):29–44. https://doi.org/10.1109/tpds.2018.2849396
2. Partheeban P, Kavitha V (2018) Versatile provisioning and workflow scheduling in WaaS under cost and deadline constraints for cloud computing. Trans Emerg Telecommun Technol 30(1). https://doi.org/10.1002/ett.3527
3. Guo W, Lin B, Chen G, Chen Y, Liang F (2018) Cost-driven scheduling for deadline-based workflow across multiple clouds. IEEE Transactions on Network and Service Management 15(4):1571–1585. https://doi.org/10.1109/tnsm.2018.2872066
4. Iyenghar P, Pulvermueller E (2018) A model-driven workflow for energy-aware scheduling analysis of IoT-enabled use cases. IEEE Internet of Things Journal 5(6):4914–4925. https://doi.org/10.1109/jiot.2018.2879746
5. Juve G, Chervenak A, Deelman E, Bharathi S, Mehta G, Vahi K (2013) Characterizing and profiling scientific workflows. Future Gen Comput Sys 29(3):682–692. https://doi.org/10.1016/j.future.2012.08.015
6. Rodriguez MA, Buyya R (2016) A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments. Concurrency and Computation: Practice and Experience 29(8). https://doi.org/10.1002/cpe.4041
7. Gupta BB, Agrawal DP (2019) Handbook of research on cloud computing and big data applications in IoT. Hershey, PA: IGI Global, Engineering Science Reference (an imprint of IGI Global). https://doi.org/10.4018/978-1-5225-8407-0
8. Gabrani N (n.d.) Formal definition of cloud computing by NIST. Retrieved from http://www.thecloudtutorial.com/nistcloudcomputingdefinition.html
9. Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I (2009) Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Gen Comput Sys 25(6):599–616. https://doi.org/10.1016/j.future.2008.12.001
10. Meena J, Kumar M, Vardhan M (2015) Efficient utilization of commodity computers in academic institutes: a cloud computing approach [Abstract]. Int J Comput Elect Autom Control Inform Eng 9(2)
11. Aloisio G, Cafaro M (2011) Scientific workflows in the cloud grids, clouds and virtualization. Springer, New York
12. Olakanmi OO, Dada A (2019) An efficient privacy-preserving approach for secure verifiable outsourced computing on untrusted platforms. Int J Cloud Appl Comput 9(2):79–98. https://doi.org/10.4018/ijcac.2019040105
13. Zhu Z, Zhang G, Li M, Liu X (2016) Evolutionary multi-objective workflow scheduling in cloud. IEEE Trans Parallel Distributed Sys 27(5):1344–1357. https://doi.org/10.1109/tpds.2015.2446459
14. Schad J, Dittrich J, Quiane-Ruiz J (2010) Runtime measurements in the cloud. Proceedings of the VLDB Endowment 3(1-2):460–471. https://doi.org/10.14778/1920841.1920902
15. Pooranian Z, Shojafar M, Abawajy JH, Abraham A (2015) An efficient meta-heuristic algorithm for grid computing. J Comb Optim 30(3):413–434

16. Fister I, Fister I, Yang X, Brest J (2013) A comprehensive review of firefly algorithms. Swarm and Evolutionary Computation 13:34–46. https://doi.org/10.1016/j.swevo.2013.06.001
17. Sousa T (2004) Particle swarm based data mining algorithms for classification tasks. Parallel Computing. https://doi.org/10.1016/s0167-8191(04)00042-0
18. Yu J, Buyya R, Tham CK (n.d.) Cost-based scheduling of scientific workflow application on utility grids. In: First international conference on e-science and grid computing (e-Science'05). https://doi.org/10.1109/e-science.2005.26ce.2005.26
19. Afzal A, Darlington J, Mcgough A (2006) QoS-constrained stochastic workflow scheduling in enterprise and scientific grids. In: 2006 7th IEEE/ACM international conference on grid computing. https://doi.org/10.1109/icgrid.2006.310991
20. Duan R, Prodan R, Fahringer T (2007) Performance and cost optimization for multiple large-scale grid workflow applications. In: Proceedings of the 2007 ACM/IEEE conference on supercomputing - SC 07. https://doi.org/10.1145/1362622.1362639
21. Garg R, Singh AK (2013) Multi-objective workflow grid scheduling using ε-fuzzy dominance sort based discrete particle swarm optimization. J Supercomput 68(2):709–732. https://doi.org/10.1007/s11227-013-1059-8
22. Smanchat S, Viriyapant K (2015) Taxonomies of workflow scheduling problem and techniques in the cloud. Future Gen Comput Sys 52:1–12. https://doi.org/10.1016/j.future.2015.04.019
23. Alkhanak EN, Lee SP, Khan SU (2015) Cost-aware challenges for workflow scheduling approaches in cloud computing environments: taxonomy and opportunities. Future Gen Comput Sys 50:3–21. https://doi.org/10.1016/j.future.2015.01.007
24. Mao M, Humphrey M (2011) Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In: Proceedings of 2011 international conference for high performance computing, networking, storage and analysis on - SC 11. https://doi.org/10.1145/2063384.2063449
25. Malawski M, Juve G, Deelman E, Nabrzyski J (2012) Cost and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds. In: 2012 international conference for high performance computing, networking, storage and analysis. https://doi.org/10.1109/sc.2012.38
26. Pietri I, Malawski M, Juve G, Deelman E, Nabrzyski J, Sakellariou R (2013) Energy-constrained provisioning for scientific workflow ensembles. In: 2013 international conference on cloud and green computing. https://doi.org/10.1109/cgc.2013.14
27. Abrishami S, Naghibzadeh M, Epema DH (2013) Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. Future Gen Comput Sys 29(1):158–169. https://doi.org/10.1016/j.future.2012.05.004
28. Calheiros RN, Buyya R (2014) Meeting deadlines of scientific workflows in public clouds with tasks replication. IEEE Trans Parallel Distrib Sys 25(7):1787–1796. https://doi.org/10.1109/tpds.2013.238
29. Poola D, Garg SK, Buyya R, Yang Y, Ramamohanarao K (2014) Robust scheduling of scientific workflows with deadline and budget constraints in clouds. In: 2014 IEEE 28th international conference on advanced information networking and applications. https://doi.org/10.1109/aina.2014.105
30. Sahni J, Vidyarthi P (2018) A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment. IEEE Trans Cloud Comput 6(1):2–18. https://doi.org/10.1109/tcc.2015.2451649
31. Chen Z, Du K, Zhan Z, Zhang J (2015) Deadline constrained cloud computing resources scheduling for cost optimization based on dynamic objective genetic algorithm. In: 2015 IEEE congress on evolutionary computation (CEC). https://doi.org/10.1109/cec.2015.7256960
32. Rodriguez MA, Buyya R (2014) Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds. IEEE Trans Cloud Comput 2(2):222–235. https://doi.org/10.1109/tcc.2014.2314655
33. Pandey S, Wu L, Guru SM, Buyya R (2010) A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In: 2010 24th IEEE international conference on advanced information networking and applications. https://doi.org/10.1109/aina.2010.31
34. Wu Z, Ni Z, Gu L, Liu X (2010) A revised discrete particle swarm optimization for cloud workflow scheduling. In: 2010 international conference on computational intelligence and security. https://doi.org/10.1109/cis.2010.46
35. Huang J (2014) The workflow task scheduling algorithm based on the GA model in the cloud computing environment. J Soft 9(4). https://doi.org/10.4304/jsw.9.4.873-880
36. Luke S (2009) Essentials of metaheuristics: a set of undergraduate lecture notes. Place of publication not identified: Lulu
37. Yang X (2008) Nature-inspired metaheuristic algorithms. Luniver Press, Frome
38. Mapetu JPB, Chen Z, Kong L (2019) Low-time complexity and low-cost binary particle swarm optimization algorithm for task scheduling and load balancing in cloud computing. Applied Intelligence 49(9):3308–3330. https://doi.org/10.1007/s10489-019-01448-x
39. Amazon Elastic Block Store (EBS) - Amazon Web Services. (n.d.). Retrieved from http://aws.amazon.com/ebs
40. Ostermann S, Iosup A, Yigitbasi N, Prodan R, Fahringer T, Epema D (2010) A performance analysis of ec2 cloud computing services for scientific computing. Cloud Computing Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, pp 115–131. https://doi.org/10.1007/978-3-642-12636-9_9
41. Anwar N, Deng H (2018) Elastic scheduling of scientific workflows under deadline constraints in cloud computing environments. Future Internet 10(1):5. https://doi.org/10.3390/fi10010005
42. WorkflowGenerator- Pegasus - Pegasus Workflow Management System. Retrieved from https://confluence.pegasus.isi.edu/
43. 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. https://doi.org/10.1109/works.2008.4723958
44. Ma T, Buyya R (2005) Critical-path and priority based algorithms for scheduling workflows with parameter sweep tasks on global grids. In: 17th international symposium on computer architecture and high-performance computing (SBAC-PAD05). https://doi.org/10.1109/cahpc.2005.22
45. Yang X (2013) Chaos-enhanced firefly algorithm with automatic parameter tuning. In: Shi Y (ed) Recent algorithms and applications in swarm intelligence research. IGI Global, Hershey, pp 125–136. https://doi.org/10.4018/978-1-4666-2479-5.ch007
46. Yang X (2009) Firefly algorithms for multimodal optimization. Stochastic Algorithms: Foundations and Applications Lecture Notes in Computer Science, pp 169–178. https://doi.org/10.1007/978-3-642-04944-6_14
47. Reddy GN, Kumar SP (2019) Regressive whale optimization for workflow scheduling in cloud computing. Int J Computat Intell Appl 18(04):1950024. https://doi.org/10.1142/s146902681950024x

Koneti Kalyan Chakravarthi is currently working as Lead Quality Engineer in Caterpillar India Pvt Ltd, Chennai, India. He received Master's degree from Jawaharlal Nehru Technological University, Anantapur, India. His research interest includes Software Engineering, Cloud Computing, and Software Quality.

Dr. L. Shyamala is currently working as Assistant Professor in Vellore Institute of Technology, Chennai, India. Her research interest includes Software Engineering, Image Processing, Cloud Computing, and Software Quality.

Dr. V. Vaidehi has done her BE in ECE from College of Engineering, Guindy, University of Madras, ME Applied Electronics from MIT and Ph.D Electronics Engineering from MIT. She has joined Madras Institute of Technology, Anna University in 1982 after serving as Scientific Assistant in I.I.Sc, Bangalore.

In MIT, She has served as Head-Computer Centre, Member Board of Studies, Member - Academic Council, Head- Electronics Engineering, Head- Computer Technology, and Head-Information Technology, Director of AU-KBC Research Center and Chairman of Faculty of Information and Communication Engineering, Anna University. She has served as Senior Professor and Dean School of Computing Science and Engineering in VIT, Chennai. Currently she is the Vice-Chancellor of Mother Teresa Women's University, Kodaikanal.

She has executed several funded research projects and published several research papers in reputed international journals and conferences. She has received several awards. Her areas of interest are Networks, Data Mining, and Image processing.