Cluster Computing (2021) 24:3449–3467
https://doi.org/10.1007/s10586-021-03314-3

Workflow scheduling of scientific workflows under simultaneous deadline and budget constraints

Ahmad Taghinezhad-Niar[1] · Saeid Pashazadeh[1] · Javid Taheri[2]

Received: 11 June 2020 / Revised: 8 May 2021 / Accepted: 27 May 2021 / Published online: 24 June 2021

Abstract
Cloud Infrastructure as a Service (IaaS) has been known as a suitable platform for the execution of workflow applications. Quality of service (QoS) in such platforms is considered a challenging problem from both customers' and service providers' perspectives to perform workflow schedules. This paper proposes Budget Deadline Delicate Cloud (BDDC) and Budget Deadline Cloud (BDC) algorithms to consider both budget and deadline constraints for scheduling scientific workflows on cloud IaaS platforms. Methods for distribution of budget and deadlines under task leveling are proposed. Four metrics (success rate, time ratio, cost ratio, and utilization rate) are utilized to evaluate the proposed algorithms' performance. Results of our proposed algorithms are compared with the BDHEFT, DBCS, and BDSD algorithms under various scenarios. Simulation results demonstrate that BDDC outperforms other algorithms in achieving cheaper costs while earning a higher success rate and utilization rate, and BDC accomplishes higher success rates and faster makespan. The performance of the proposed methods is confirmed using a real cloud environment.

Keywords Scheduling · Workflow applications · Deadline · Budget · Quality of services

1 Introduction

Speed of data transferring and calculation has become essential in human life; therefore, the need for fast and easy processing has been emphasized even more. Distributed systems, especially cloud computing, emerged as a successful response of essence. Cloud computing mainly supports three types of services: Platform as a service, Infrastructure as a service (IaaS), and Software as a Service [10]. Nowadays, cloud computing is better-known to present on-demand models of computation and storage services. Therefore, it causes an ongoing migration from high-performance computing systems to the cloud.

One of the vital challenges of cloud computing is performance under-utilization, which causes costly problems. Cloud providers usually present hourly charging computation services that are also known as the pay-per-use charging model. They offer IaaS charge-based services like CPU, memory, storage, and bandwidth usage. Elasticity and cost models of cloud computing cause economic challenges from clients' and cloud provider's perspectives [15].

Cloud provides a better capacity to automate workflows and execute complex computation-intensive and data-intensive tasks produced by scientific experiments. This capability must follow an appropriate resource provisioning and schedule plan to avoid costly penalties and inefficient utilization; these are substantial cloud challenges. Moreover, financial problems caused by disorganized schedules arise, especially in dynamic cloud markets like the Amazon spot market [8].

Different QoS issues for workflow schedules are the objective of many research endeavors [26, 30]. Most of the presented solutions consider a single objective like

✉ Saeid Pashazadeh
pashazadeh@tabrizu.ac.ir

Ahmad Taghinezhad-Niar
a.taghinezhad@tabrizu.ac.ir

Javid Taheri
javid.taheri@kau.se

[1] Faculty of Electrical and Computer Engineering, University of Tabriz, Tabriz, Iran

[2] Department of Mathematics and Computer Science, Karlstad University, Karlstad, Sweden

makespan [26] or cost [31]. Therefore, the need for satisfying various parameters of QoS in workflow scheduling has become inevitable. Knowing that even a single objective scheduling problem is NP-complete [12, 13], considering multiple constraints such as budget and time over an unlimited heterogeneous set of resources exposes even more challenging situations. Thus, considering 'Cost' and 'Time' as two conflicting constraints simultaneously is very challenging. The time complexity of the stated problem demands a long time to acquire any sensible schedule such as decreasing makespan and provisioning of expensive resource instances; therefore, designing efficient heuristics is necessary [8].

A limited number of works such as DBCS, BDHEFT, and BDSD have considered both budget and cost constraints simultaneously. DBCS and BDSD evaluate the Planning success rate, which is the probability of a schedule plan under their demanded cost and time. BDHEFT evaluates the normalized cost and time of the schedule plan with other algorithms. Hence, algorithms to satisfy both budget and deadline constraints while proposing a higher quality of services (faster time and cheaper cost) with different preferences are still needed.

To this end, we present Budget Deadline Delicate Cloud (BDDC) and Budget Deadline Cloud (BDC) algorithms for IaaS cloud. They are constrained by both budget and deadline and propose a higher QoS to decrease the schedule plan's time and cost to provide a higher utilization rate. BDDC and BDC present novel deadline and budget distribution methods and propose a trade-off factor using these distribution methods to provide suitable schedule plans to address the mentioned challenges.

BDDC aims to achieve a higher success rate while acquiring an economical (cheaper) schedule plan, and BDC aims to provide a higher success rate while providing a faster time (makespan) for the schedule plan. To demonstrate our contributions, we have evaluated the presented algorithms with four evaluation metrics (success rate, time ratio, cost ratio, and utilization rate) under different constraints from strict to loose with state-of-the-art algorithms to schedule different scientific workflows.

Recently, the public cloud added a new service as a function as a service (FaaS) for serverless computing. In FaaS, the user uploads its tasks on the cloud and defines its memory requirements. The analysis in [20] illustrates that computation-intensive tasks are more suitable to execute in FaaS than data-intensive tasks. Our proposed algorithms are also applicable for workflow scheduling in the FaaS with modifications in resource definition where resources are defined as function configurations. Each function configuration presents a memory size and requires a monetary cost. In FaaS, each task is assigned to a function configuration instead of cloud instances. Contributions of this paper are as follows:

- we proposed two novel algorithms called BDDC and BDC for workflow scheduling under budget-deadline constrained to meet-up with the preference of QoS (Time vs Cost);
- we performed a comprehensive evaluation of our presented algorithms to demonstrate the time and cost-efficacy, success, and utilization rate of schedule plan under different circumstances; and
- we implemented our proposed algorithms on a real system to showcase their efficiency for real deployments.

We organized this article as follows. Section 3 presents the related works. Section 4 describes the problem description. Section 5 articulates our proposed algorithms and methods of budget and deadline distribution. Section 6 articulates the time complexity of the proposed algorithms. Section 7 illustrates the BDDC and BDC algorithms' results and effectiveness with comparable recently published works. Finally, Sect. 8 summarizes the paper and highlights our future plans.

2 Applications with budget or deadline constraints

Many large-scale scientific applications in various subjects such as climate and medical modeling, simulations in business continuity, and disaster recovery are deadline-constrained applications [17]. Deadline-sensitive applications are categorized into hard and soft deadline-constrained applications. In applications with hard deadline constrained (e.g., air traffic control applications and railway signaling systems), achieving results before the demanded time is crucial. In soft deadline constrained applications, missing a deadline is not extremely essential, however, it causes undesirable impacts on the system's performance; typical examples are weather prediction workflow and health tracker applications.

Budget is one of the main concerns for clients of pay-per-use services in the cloud [18]. Scientific institutes and digital content makers like video encoding industries wish to execute their workflow applications within the desired time and budget. Parameter sweeping applications (e.g., financial applications) also generally use cloud services to find optimal answers. Even in scientific computation, some projects can be considered a parameter sweeping application, for example, when searching for Extra-Terrestrial Intelligence to detect intelligent life outside Earth by analyzing the radio frequency signals coming from space [5]. To this end, some cloud providers such as Amazon Web

Services (AWS) and Google Cloud have already presented preliminary (i.e., not necessarily optimal) tools and tutorials to manage user budgets in the provided services [22].

3 Related works

Scheduling problems are investigated for two different types of tasks: (1) independent and (2) dependent workflow tasks. Workflow tasks are represented by a directed acyclic graph (DAG). A task can be executed after their precedence tasks finished their execution and the task's input data become available.

The main difference between Cloud and Grid systems is the pricing model in which resources are leased based on time intervals on the cloud. Hence, the cloud scheduling problem is even more complicated. Therefore, in addition to task selection and allocation on the cloud environment's schedule plan, a resource provisioning phase is also needed. Many papers consider scheduling problems based on the Grid system; meanwhile, a limited number of works consider it with the cloud pricing model.

Scheduling problems are solved using three types of algorithms: meta-heuristic, heuristic, and hybrid. In workflow scheduling, well-known meta-heuristics such as Partial Swarm Optimization and Genetic Algorithm are used frequently [12, 13]. However, the difficulty of adapting the execution phase changes and higher time complexity to provide an appropriate schedule plan make meta-heuristics less effective in real environments. Thus, the use of heuristic algorithms is more common [6].

Most of the workflow scheduling methods are investigated by single objective scheduling algorithms [4, 12, 29, 30, 33]. However, since cloud computing has become popular, providing a better QoS among cloud providers became a priority. Therefore, recent works consider QoS as a multi-objective schedule problem. Objectives such as cost [7], time, energy, and security [1] are among the most considered QoS objectives. Among them, cost and time for public clouds are deemed more important for most industries and researches [32].

Some researchers considered budget-constrained workflows like [7, 18] and some considered deadline-constrained workflows [11]. In the deadline-constrained algorithms, the objective is to reduce cost as much as possible to meet the desired deadline. Thus, deadline distribution heuristics such as Deadline Bottom level (DBL) [29] algorithm are proposed for achieving a faster time under a given budget in budget-constrained workflows [6, 8, 25, 27].

One of the primary works that address workflow scheduling in distributed systems is the heterogeneous earliest finish time (HEFT) algorithm presented by Xie et al. [26]. The main idea of HEFT is partially used to present new algorithms in [6, 18, 27, 31]. HEFT is a two-phase algorithm. In the first phase, it prioritizes tasks to determine critical paths based on computation and communication time. In the second phase, tasks are selected based on their rank. Each of them is allocated to the CPU with the lowest execution time among all CPUs.

In budget-constrained algorithms, Ghafouri et al. [18] proposed an algorithm for workflow scheduling to reduce schedule time (makespan) as much as possible with the demanded cost. They try to schedule critical paths first and then schedule noncritical paths using the back-tracking technique. Arabnejad et al. [7] presented a few budget distribution strategies over workflow tasks and schedule them under the given budget while reducing the makespan. First, tasks are distributed over different levels using the DBL algorithm. Then, they evaluate different budget distribution strategies like the number of tasks on each level, the number of tasks through the graph's exit task, and a method called All-in. All-in considers the whole budget for the first level, and after consuming the budget in each level drops the rest of it to the following levels, which are not currently scheduled. They have concluded that the All-in technique had the fastest results among their presented strategies. Thus, All-in is utilized for the budget distribution phase in their other work [9].

Chakravarthi et al. [14] proposed a budget-constrained workflow scheduling algorithm for IaaS clouds. Their algorithm was based on normalization methods to provide a reasonable makespan under the user demanded budget. They utilized scientific workflows to evaluate the performance of their proposed algorithms.

In deadline-constrained algorithms, Zheng et al. [30] proposed three novel heuristic algorithms to reduce scheduling cost as they satisfy the deadline for distributed systems. They were trying to reduce the cost of scheduling in multicore resources by allocating tasks to the cores with lower frequency while making sure that the task's deadline is not exceeded. Their algorithm builds a schedule plan based on the HEFT [26] algorithm in the first phase. Consequently, they changed the task's core to lower frequencies to save monetary cost until the makespan reaches the edge of its deadline. Abrishami et al. [2] presented two deadline-constrained scheduling algorithms. Their scheduling model is based on the cloud IaaS model in which instances of different resource types are provisioned before task execution. Their algorithms find critical paths recursively from bottom to top of the workflow, and then schedule the path with the cheapest resource instance that can meet the deadline.

Another deadline-constrained algorithm was presented by Yuan et al. [29]. They named it DBL; it divides tasks into some levels based on their distance to the exit task so

that tasks of a level do not have precedent constraints. Then, DBL distributes tasks on the levels based on their time intervals. The DBL algorithm has a low complexity solution to divide the tasks of a workflow into the different levels in which tasks of a level have not precedence constraints within each other [29].

In [3] a workflow scheduling deadline-constrained algorithm for big-data applications on the cloud is presented. Their method investigates resources to find an execution host for workflow tasks with minimum monetary cost while still satisfying a deadline. They utilized Montage and randomly generated workflows to evaluate the performance of their work.

In budget-deadline constrained algorithms, Arabnejad et al. [8] proposed a budget-deadline aware scheduling (BDAS) heuristic for workflow scheduling under budget-deadline constrained. First, they used DBL [29] algorithm to level the tasks. Then, they set a sub-deadline for each level considering a task's number multiplied by the maximum value of the earliest completion time (ECT) among all tasks of the same level. Finally, a trade-off factor similar to [6] is utilized to balance time and cost. Their deadline distribution method leads to setting an unbalanced sub-deadline on the level where ECT of level tasks and the number of tasks have diversity, especially on the neighbor levels. For example, there could be a level that contains ten tasks. One of them requires 15-minute execution and 9 of them need less than 1-minute of execution time. Their deadline distribution algorithm distributes the demanded deadline based on a formula that comes from the maximum execution time of the level's tasks multiplied by the task number. Consequently, it leads to inappropriate restriction of the heuristic to meet its objectives.

BDHEFT algorithm is a cost and time efficient algorithm which is proposed by Verma et al. [27]. Their algorithm is an extension to the HEFT algorithm [26] that considers both cost and time as two objectives in a pay-per-use cloud environment. It calculates a spare budget and a spare deadline for the workflow and a spare budget and deadline for each task in each resource provisioning step. Then, a trade-off value (with the task's earliest finish time and earliest execution cost) is calculated to find the best possible resource. They evaluated BDHEFT using scientific workflows by two performance metrics: normalized schedule cost and normalized schedule length. Former indicates the workflow cost proportion to the cheapest cost, and the latter denotes the proportion of makespan to the fastest possible execution time. The BDHEFT algorithm is extended to adapt the cloud pricing model by an algorithm presented for the grid environment [31]. BDHEFT, however, does not consider budget and deadline constraints. Therefore, users can not define constraints and customize the importance of budget or deadline.

Arabnejad et al. [6] presented DBCS, a budget-deadline constrained algorithm for heterogeneous systems. DBCS used the HEFT approach in the task selection phase. Then, in the resource selection, they proposed a trade-off factor to find the task's proper resource. For each task, the candidate resources are selected from the resources that cost less than the cheapest resource and spare budget aggregation. The spare budget defines the difference between the remaining budget and the cheapest possible cost of unscheduled tasks. They used task sub-deadline in the trade-off factor, which is calculated based on the tasks' aggregation time that constitutes a critical path from the task to the exit node. The DBCS algorithm does not consider the pay-per-use cloud cost model.

A low-time complexity algorithm named BDSD is proposed by Sun et al. [25]. They calculated a sub-deadline for each task based on the critical path from the task to the exit task. Then, they defined a factor, which is the proportion of the remaining budget to the number of unscheduled tasks, to limit the number of resources in the resource allocation phase. In the resource allocation step, resources are limited to those whose monetary cost is less than the proportion of the remaining budget to the number of unscheduled tasks. Therefore, when there is a high disparity in a workflow tasks' runtime, it could lead to an unbalanced schedule plan (exceeding deadlines or budget). Meanwhile, scientific workflows like Cybershake, Ligo, and Epigenomic have different tasks with very heavy runtime variance.

In this paper, a cloud cost model is considered suitable for a public cloud with dynamic instance provisioning to schedule workflow applications. They can be utilized for private, hybrid, and public clouds with a little customization. Additionally, Two novel algorithms are proposed with user preference in monetary cost at BDDC algorithm and time preference in BDC algorithm for simultaneous budget and deadline constraints.

For the evaluation of our algorithms, we will compare them with the state-of-the-art algorithms, namely BDHEFT [27], DBCS [6], and BDSD [25]. Four evaluation metrics are selected to demonstrate the time and cost-efficacy of all algorithms.

4 Problem description

4.1 Application model

Distributed computations are represented by a DAG workflow model. A DAG defines workflow as $G = (T, E)$ where $T = \{t_0, t_1, \ldots, t_n\}$ is set of task collection and E represent tasks' relationship and order. $E = \{e_{(i,j)} | (t_i, t_j \in$

$T)\}$ indicates a set of directed acyclic edges indicating the precedence order and data transfer of tasks. $e_{(i,j)}$ indicates that task t_i is a parent of task t_j, that is, the output data of t_i will be used as input data to t_j. Tasks could have multiple parents or precedent multiple tasks. t_i can not be executed unless all of its parent tasks are finished, and their data are available.

4.2 System model

IaaS service model of the cloud is adopted in this article. There are specific resource types with different memory, CPU, storage, and network bandwidth; all can be leased as an instance for demanded time intervals. Time interval (TI) is mostly based on one hour. This paper uses the resource model of Amazon Elastic Compute Cloud (Amazon EC2). It offers services where instances are provisioned with a specific type of resource with a pay-per-use price model, which is based on an hourly time interval. Let $R = \{r_1, r_2, \ldots, r_m\}$ be the different resource types offered by cloud providers. Each resource has a different price for a time interval. An instance is provisioned with a resource type, which is denoted by $ins_{i,k}$ where i indicates the instance index and k indicates the resource type index. All instances and storage spaces are assumed to be on an identical region and bandwidth. Hence, the price of data transferring is zero.

4.3 Problem definition

Because the task execution time is already known, we used the task runtime estimation technique from [16, 21, 24]. Efficient scheduling algorithms require high precision in predicting the execution time of tasks. Two most recent papers are proposed to alleviate this problem [21, 24]. In [24], Suh et al. proposed a clustering-based scheme in runtime estimation for scientific simulations. In [21], Kim et al. proposed a runtime estimation scheme for high-performance computing using machine learning with 73% accuracy in runtime estimation for real simulation data.

In this paper, we used real scientific workflow tasks, which are common for evaluating workflow scheduling algorithms [6, 8, 18, 25, 27, 30]. We accumulated scientific workflows that are generated by tracing real scientific applications data [19]. Workflow execution finish time is called *makespan*, which is defined by the exit task finish time (FT) in Eq. (1).

$$makespan = FT(t_{exit}) \tag{1}$$

Transfer time from t_i to t_j defined in Eq. (2). Transfer time between t_i and t_j is negligible when they are on an identical instance. BW indicates the network bandwidth. $\left|data_{t_i,t_j}\right|$ is the amount of data that need to be transferred between t_i and t_j, and ins_j denotes the instance j.

$$TF_{i,j} = \begin{cases} \dfrac{\left|data_{t_i,t_j}\right|}{BW}, & ins_i \neq ins_j \\ 0, & ins_i = ins_j. \end{cases} \tag{2}$$

Equation (3) defines the completion time of a task, where $w_{t_i}^{min}$ indicates the fastest time for computation of the task among the resources. $pred(t_i)$ determines the sets of immediate predecessors of the task t_i.

$$cpTime(t_i) = w_{t_i}^{\min} + \max_{t_p \in pred(t_i)} \left\{TF_{p,i}\right\} \tag{3}$$

The Earliest Start Time (EST) of a task is calculated based on the fastest resource type, which is represented in Eq. (4).

$$EST_{t_i} = \begin{cases} \max\limits_{t_p \in pred(t_i)} \left\{EST_{t_p} + cpTime(t_p)\right\}, & otherwise \\ 0, & t_i = t_{entry}. \end{cases} \tag{4}$$

If the time interval is one hour, even using 1 min of the instance will cost the whole hour. Equation (5) defines the billing cycle (BC) of an instance, which is the end of the instance time interval. TI indicates the time interval (i.e., one hour).

$$BC_{ins_j} = \left\lceil \frac{FT(ins_j)}{\mathrm{TI}} \right\rceil \tag{5}$$

$FT(ins_j)$ indicates the finish time of ins_j. $TC_{t_i}^{ins_j}$ denotes the Task Cost (TC) for executing t_i on ins_j, which is defined in Eq. (6) and $FT_{t_i}^{ins_j}$ indicates t_i's finish time on instance ins_j. $price_{ins_j}$ denotes the price of ins_j for a time interval.

$$TC_{t_i}^{ins_j} = \begin{cases} \left\lceil \dfrac{FT_{t_i}^{ins_j}}{\mathrm{TI}} \right\rceil * price_{ins_j}, & otherwise \\ 0, & \left\lceil \dfrac{FT_{t_i}^{ins_j}}{\mathrm{TI}} \right\rceil \leq BC_{ins_j}. \end{cases} \tag{6}$$

Let assume DD and DB as the user demanded deadline and budget, respectively, for a workflow application. The purpose is to discover a schedule plan that the finish time of its exit task is less than DD and the monitory cost of the schedule plan is less than DB for a given workflow. The main scheduling problem is formulated in Eq. (7). Equation 7b ensures that every task t_i is scheduled on an instance. Equation 7c indicates that the finish time of the exit task (makespan) must be less or equal to a demanded deadline. Equation 7d ensures that executing tasks' total monetary cost is less than the demanded budget.

$$\theta_i^j = \begin{cases} 1, & t_i \text{scheduled on } ins_j \\ 0, & otherwise \end{cases}$$

$$(\text{Problem})\theta_i^j \in \{0,1\} \tag{7a}$$

$$\sum_{i=1}^{n} \theta_i^j = 1, j = 1...m \tag{7b}$$

$$FT(t_{exit}) <= DD \tag{7c}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m} \theta_i^j . TC_{t_i}^{ins_j} <= DC \tag{7d}$$

5 Proposed algorithms

In this section, two budget-deadline constrained algorithms, namely BDDC and BDC, are proposed aiming to find a solution for the problem described in the previous section. BDDC is a heuristic algorithm that distributes the budget and deadlines for each level considering the entire task's completion time to acquire economic costs for each level. Meanwhile, BDC distributes deadlines to each level, and the budget is considered as the remaining cost at each scheduling cycle. Therefore, BDC does not utilize the BudgetDistribution algorithm and uses *remainingBudget* as the sub-budget for each task. However, the rest of the steps of BDC are similar to those of the BDDC algorithm.

5.1 BDDC algorithm

The presented BDDC algorithm contains five main stages: (1) the workflow task leveling, (2) the DeadlineDistribution algorithm, (3) the BudgetDistribution algorithm, (4) the task selection, and (5) instance provisioning.

5.2 Task leveling

In this stage, the tasks are divided into different levels based on the DBL algorithm. Tasks of a level must not have precedent constraints within each other.

In this paper, we use the DBL task leveling method whereby the Bottom Depth (*BD*) of each task determines its level-id. Henceforth, we add two dummy tasks as an entry task and an exit task to each DAG; execution time and cost for both tasks are zero. The *BD* of a task is determined by the maximum number of tasks from it to the exit task as calculated by Eq. (8).

$$BD_{t_i} = \begin{cases} \max\limits_{t_s \in succ(t_i)} \{BD_{t_s} + 1\}, & otherwise \\ 0, & t_i = t_{exit}. \end{cases} \tag{8}$$

where $succ(t_i)$ determines the sets of immediate successors of the task t_i. Figure 1 depicts a workflow DAG and a set of tasks with their BD levels ($level_i$).

5.3 Distribution of deadline

After dividing the tasks into different levels, a novel distribution of deadline method is applied. First, we calculate a $\sigma weight$ in Eq. (9) that is the accumulation of total tasks estimated completion time.

$$\sigma weight = \sum_{i=1}^{n} cpTime(t_i) \tag{9}$$

Afterward, the level's weight (LW_k) is calculated for each level with the unscheduled tasks of that level; that is defined in Eq. (10). $level_k$ indicates the set of tasks that are divided into the $level_k$; *UnSch* indicates the set of tasks that have not been scheduled yet.

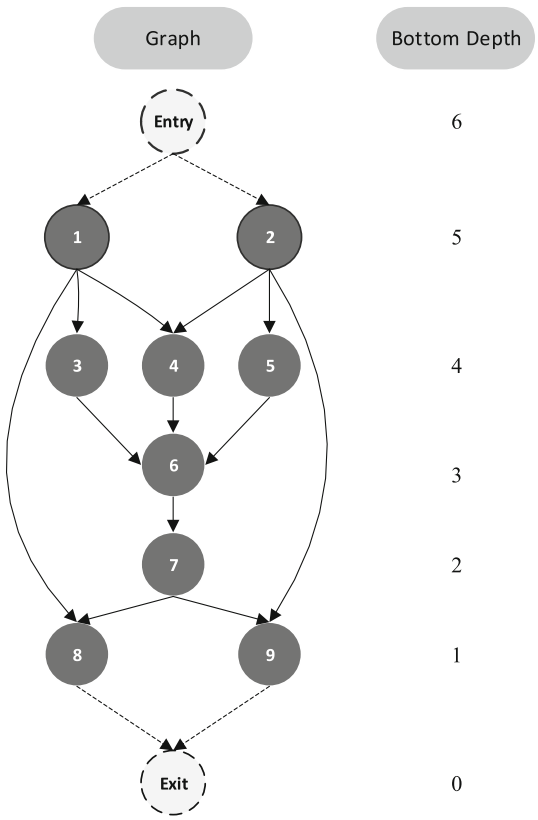


Fig. 1 DAG of workflow with their bottom depth (level id)

$$LW_k = \sum_{t_i \in \text{levelTasks \& UnSche}}^{|\text{levelTasks}|} cpTime(t_i) \tag{10}$$

Then sub-deadline (*SDL*) is calculated for each level and is defined in Eq. (11).

$$SDL_k = SDL_{k-1} + \left(LW_k * \frac{DD}{\sigma weight} \right) \tag{11}$$

In this way, every task completion time is a contributor to calculate *SDL* of its level. It plays a notable role in acquiring a balanced time-cost heuristic schedule in the instance provisioning stage.

5.4 Distribution of budget

The BDDC algorithm distributes budget on each level based on Eq. (12), where SBL_k represents the sub-budget of $level_k$. *remainingBudget* is the difference between the current Demanded Budget (DB) and the consumed budget. Algorithm 1 illustrates the distribution of demanded budget over the workflow tasks.

$$SBL_k = SBL_{k-1} + \left(LW_k * \frac{remainingBudget}{\sigma weight} \right) \tag{12}$$

Algorithm 1: BudgetDistribution

```
1 Calculate σweight from Eq. (9);
2 σBudget ← remainingBudget / σweight;
3 foreach level_k : levelsSet do
4 |   LW_k ← 0 ;
5 |   foreach Tasks ∈ (level_k and UnSch) do
6 |   |   Calculate LW_k from EQ.(10)
7 |   end
8 |   SBL_k ← LW_k * σBudget;
9 end
```

5.5 Task distribution

In this stage, tasks are selected to send to the instance provisioning algorithm one-by-one, starting from the entry task's children. We considered *EST* of ready tasks as a priority. Ready tasks are the tasks that all of their parent tasks are currently scheduled. Algorithm 2 illustrates the task distribution phase where each task is sent to the instance provisioning algorithm to be scheduled. In line 1, children of entry Task (a dummy task) are added to Q, which indicates the list of ready tasks. Then, the tasks of Q are sorted by their *EST* in the ascending order to be sent for the instance provisioning algorithm.

Algorithm 2: TaskDistribution

```
1  Q ← G.entryTask.Childeren;
2  while Q is not empty do
3  |   task ← minEST(Q) ;
4  |   InstanceProvisioning(task);
5  |   foreach child_i : task.Childeren do
6  |   |   if all parents of child_i are scheduled then
7  |   |   |   Q ← Q ∪ {child_i}
8  |   |   end
9  |   end
10 |   Q ← Q − {task}
11 end
```

5.6 Instance provisioning

In this step, instances are provisioned from the resource types, and tasks are allocated to them. Two expressions are presented to establish a trade-off value between time and cost. Time quality (TQ) is defined in Eq. (13) to determine a trade-off value for the execution time of the task on an instance. Cost Quality (CQ) defined in Eq. (14) represents a trade-off value for the cost of running the task on an instance. Both TQ and CQ return a negative value when the time exceeds the sub-deadline and cost exceeds a task's sub-budget. SDL_k denotes the sub-deadline and SBL_k denotes the sub-budget of the level whereby t_i belongs to it.

$$TQ_{t_i,ins_j} = \frac{SDL_k - FT_{t_i}^{ins_j} - FT_{t_i}^{min}}{SDL_k} \quad , t_i \in level_k \tag{13}$$

The min superscript of *FT* denotes the minimum finish time of a task among all instances. In *TaskCost*, it represents the minimum cost for executing a task among all instances.

$$CQ_{t_i,ins_j} = \frac{SBL_k - TC_{t_i}^{ins_j} - TC_{t_i}^{min})}{SBL_k} \quad , t_i \in level_k \tag{14}$$

Afterward, $Quality_{t_i,ins_j}$ is defined by Eq. (15); it is calculated using *CQ* and *TQ*. ω is a weight parameter to control the impact of the time to cost of the schedule plan. Algorithm 3 illustrates the instance provisioning stage. First, a task's *Quality* is examined among the already provisioned instances, and then it seeks the resource types (line: 2–15). Suppose the maximum *Quality* comes from one of the already existed instances. In that case, the task will be allocated to that instance. Otherwise, if it comes from one of the other resource types, a new instance of that resource type will be provisioned. The task is then scheduled to the newly provisioned instance. If the max value of *Quality* is less than 0, it means that the appropriate instance is not found with the calculated sub-deadline or sub-budget. Hence, it recalculated sub-deadline and sub-budget for just one more time with the unscheduled tasks using the methods presented in Sects. 5.3 and 5.4, respectively. Afterward, it goes to Line 2 to continue the instance

provisioning algorithm (line: 16–20). For the BDC algorithm, only the DeadlineDistribution algorithm is called. If the max value of *Quality* belongs to a resource type; first, an instance of the resource type will be provisioned (line: 21–24). Second, the task will be allocated to the instance with the maximum *Quality* value.

$$Quality_{t_i,ins_j} = \omega * TQ_{t_i,ins_j} + (1-\omega) * CQ_{t_i,ins_j} \quad (15)$$

Algorithm 4 illustrates the BDDC algorithm. At first, the user demanded deadline and budget are inserted, then the task leveling algorithm runs, and levels are created. Afterward, the Distributing deadline and budget algorithms are executed. Consequently, task distribution sends each task based on its priority to the instance provisioning algorithm. Conclusively, the algorithm returns the schedule plan. Regarding Algorithm 4, the BDC algorithm does not have the BudgetDistribution algorithm; and the sub-deadline (*SBL*) of each task is derived from the remaining budget.

Algorithm 3: InstanceProvisioning(task)

```
1  max ← −∞; mres ← null; repeat ← 0 ;
2  foreach ins_j : instanceSet do
3  |  calculate Quality_{task,ins_j} from Eq.(15) ;
4  |  if max < Quality_{task,ins_j} then
5  |  |  max ← Quality_{task,ins_j} ;
6  |  |  mres ← ins_j;
7  |  end
8  end
9  foreach res_k : resourceSet do
10 |  calculate Quality_{task,res_k} ;
11 |  if max < Quality_{task,res_k} then
12 |  |  max ← Quality_{task,res_k} ;
13 |  |  mres ← res_k;
14 |  end
15 end
16 if max < 0 and repeat < 1 then
17 |  DeadlineDistribution() ;
18 |  BudgetDistribution() ;
19 |  repeat ← 1 ;
20 |  go to line 2 ;
21 end
22 if mres ∈ resourcesSet then
23 |  create new instance from mres ;
24 |  instanceSet ← instanceSet ∪ {mres} ;
25 end
26 allocate task to the instance that mres refers to ;
27 set task as a scheduled task;
```

Algorithm 4: BDDC algorithm

```
1 DD and DC are initiated by the user ;
2 G ← Workflow DAG ;
3 levels← Task leveling(G) ;
4 DeadlineDistribution(levels) ;
5 BudgetDistribution(levels) ;
6 TaskDistribution(G) ;
7 return the schedule plan;
```

6 Time complexity analysis

To compute the time complexity of our algorithms, we consider a DAG $G = (T, E)$ where T indicates the tasks and E indicates the dependency among them. If we consider tasks number as n, then the maximum dependency of tasks is calculated by $\frac{(n(n-1))}{2}$. Processing tasks *EST*, *FT*, and *TF* require a time complexity of $O(n^2)$ which are essential for decision making for planning tasks [6, 25, 27]. The workflow partitioning stage needs a time complexity of $O(T+E)$, equal to $O(n^2)$ in the worst-case scenario. Afterward, the DeadlineDistribution algorithm encompasses (1) calculating $\sigma weight$, (2) leveling weights, and (3) setting sub-deadlines for each level; each stage has $O(n)$ time complexity. Ultimately, the DeadlineDistribution algorithm needs time complexity equal to $O(3n)$, likewise the BudgetDistribution algorithm. Additionally, the DeadlineDistribution and BudgetDistribution algorithms can be calculated at once; therefore, the time complexity of them is $O(n)$. The instance provisioning stage needs time complexity equal to $O(n.m)$ where m indicates the instances number (at most n). Moreover, if the resource provisioning stages need a redistribution of deadlines and budget, the time complexity increases to $O(2(n.m)+n)$. To summarize, the overall time complexity of both presented algorithms is $O(n^2)+O(n)+O(2(n.m)+n)$; as a result, both of the algorithms have time complexity with the order of $O(n^2)$; this is known as acceptable for scheduling workflow tasks on heterogeneous resources [6, 25, 27]. Moreover, in our experiments, the redistribution of budget and deadline in the instance provisioning stage only happened for 1% of n tasks.

7 Evaluation

This section presents the performance of our BDDC and BDC algorithms against other algorithms, namely BDSD [25], DBCS [27], and BDHEFT [6]. The experiments contain two sections: (1) empirical analysis and (2) simulation. Due to the accessibility of limited resources, a small

Table 1 Resource types of Amazon EC2 cloud

Resource type	Memory (GB)	ECU	Cost/hour
m5.large	8	10	$0.096
m5.xlarge	16	16	$0.192
m5.2xlarge	32	37	$0.384
m5.4xlarge	70	64	$0.768
m5.8xlarge	128	128	$1.536
m5.12xlarge	192	168	$2.304

number of workflow tasks are evaluated in a real cloud environment. However, simulation experiments are included to generate a comprehensive analysis of larger workflows. We have simulated a single data center of Amazon EC2 with six resource types detailed in Table 1. The resource-type configuration is based on the recent on-demand services provided by US-East (Ohio) Amazon EC2. AWS provides a traditional Import/Export mechanism from its high-speed internal network to facilitate low-speed transferring data to become online for data-intensive workloads. Thus, it is a feasible service to execute a large volume of data-like scientific workflows [23]. AWS is used in profiling scientific workflows in [19]. It is worth noting that Amazon EC2 services are used as a case study, and the provided algorithms are not dedicated to any particular cloud provider service.

Scientific workflows are real applications that are commonly used in the compared methods [6, 25, 27] and other state-of-the-art workflow scheduling algorithms [12, 13]. They are characterized and profiled in [19] as synthetic workflows to improve the evaluation of scheduling algorithms [27]. We have chosen five different workflows commonly used in various researches that contain Input/output (I/O) intensive and computation-intensive applications with different shapes of the graph.

The Montage workflow is an engine to create mosaics for astronomical images for NASA space; it is an I/O-intensive workflow. A Ligo workflow is used to detect gravitational waves. It is known to be a computation-intensive workflow. A Sipht workflow is from Harward university's bioinformatics project for automation of search in untranslated sRNA in the database of National Center. It is a computation-intensive workflow. Epigenomics is a computation-intensive workflow to automate different operations in genome sequence processing. The Cybershake is a data-intensive workflow to analyze earthquake risks [19].

The network bandwidth (BW) is fixed to 20 MBps, like [6, 25, 27]. Our presented algorithms are evaluated with common real-world scientific DAGs, such as Montage, Epigenomics, Sipht, Cybershake, and Ligo (Inspiral) under various constraints. Various constraints of deadline and budget (from strict to loose) are selected for an extensive evaluation of our work. We have considered an hourly time interval according to the Amazon EC2 time interval. To calculate various constraints in our experiment, we defined two variables as $minTime_{cp}$ in Eq. (16) and $maxTime_{cp}$ in Eq. (17). cp indicates the list of tasks in the critical path of the workflow.

$$minTime_{cp} = \sum_{t_i \in cp} (cpTime(t_i)) \tag{16}$$

$$maxTime_{cp} = w_{t_i}^{\max} + \max_{t_p \in pred(t_i)} \{TF_{p,i}\} \tag{17}$$

Demanded Budget constraint (DB) is defined by Eq. (18) as it is used in [6, 25], and the Demanded Deadline constraint (DD) is defined by Eq. (19) as it is used in [6]. The range of f_b and f_d is selected by [0, 1] as in [6, 25]. $maxCost$ and $minCost$ are the maximum and minimum likely costs for running the workflow DAG; they are acquired by summing the highest and lowest execution cost of each task among all the instances, respectively. The budget constraints range and deadline constraint range in the experimental results are calculated by Eq. (18) and Eq. (19). In Figs. 5, 6, 7, 8 and 9, when a deadline is fixed (i.e., 0.1), the budget constraints are variable (i.e., [0.1, 0.6]), and when a budget is fixed, deadline constraints are variable.

$$DB = minCost + f_b * (maxCost - minCost) \tag{18}$$

$$DD = minTime_{cp} + f_d * (maxTime_{cp} - minTime_{cp}) \tag{19}$$

The first row of Table 2 displays the final central system that is used in our experiments (Sect. 7.2). We have used five common real scientific workflows under various types of constraints to evaluate algorithms in which a sample structure of them is depicted in Fig. 4 [19]. We use four performance evaluation metrics (success rate, time ratio, cost ratio, and utilization rate) to evaluate our algorithms' performance.

The *SuccessRate* is the probability of acquiring a schedule plan satisfying the demanded budget and deadline constraints. Several papers use normalized cost and normalized time to demonstrate time and cost efficiency [27]. However, we have utilized time ratio and cost ratio to evaluate the efficacy of time and cost of the algorithms;

Table 2 Configuration of desktop systems used in the empirical experiments

Cpu type	Freq	Cores	Memory (GB)	OS	Cost
Intel Core i7-3630	2.4	4	8	Win10	1
Intel Core i3-2040	2.0	2	4	Win10	0.5

this is based on the suggestion provided in [9]. The *SuccessRate*, which is a common metric to evaluate the performance of budget-constrained or deadline-constrained algorithms, is defined in Eq. (20) [6, 9, 25].

$$SuccessRate = \frac{|\text{Successful Experiments}| * 100}{|\text{Experiments}|} \tag{20}$$

The cost ratio is defined in Eq. (21); it is the proportion of acquired *ScheduleCost* and demanded budget (DB). Time ratio [9] is acquired by the proportion of *makespan* to the demanded deadline (DD) and is defined in Eq. (22).

$$CostRatio = \frac{ScheduleCost}{DB} \tag{21}$$

$$TimeRatio = \frac{makespan}{DD} \tag{22}$$

Results for different deadline constraints acquired with constant f_b and variable f_d with the range of [0.1, 0.6] and the increment of 0.1, respectively. Results for various budget constraints acquired with constant f_d. Hence, experimental results for time ratio, cost ratio, success rate, and mean Utilization Rate (UR) for each workflow are acquired with 36 runs with different constraints.

Last but not least, the mean UR is a metric to demonstrate the performance utilization of the resources and their idle time. *UR* is significant to provide an economic schedule plan. EX_{ins_j} defined in Eq. (23) is the sum of execution time (runtime) of all tasks that are executed in ins_j. The UR of an instance is defined in Eq. (24).

$$EX_{ins_j} = \sum_{t_i \in ins_j.tasks} w_{t_i}^{ins_j} \tag{23}$$

$$UR_{ins_j} = \frac{EX_{ins_j}}{FT(ins_j) - ST(ins_j)} \tag{24}$$

7.1 Empirical performance analysis

Cloud providers such as Amazon EC2 and Microsoft Azure do not allow end-users to manipulate scheduling policies. Therefore, researchers [7, 9, 25, 28] use simulation tools such as Cloudsim to evaluation their scheduling algorithms. Furthermore, most platforms in the market do not have user-based scheduling policies. Aneka is a PaaS platform that, unlike others is developed to present various programming models and infrastructures for cloud computing. Therefore, we used Aneka to confirm the results of our scheduling algorithm for the real cloud environment.

7.1.1 Aneka platform

In this paper, Aneka platform version 5.0 [5] and Microsoft Visual Studio 2019 are utilized to configure six desktop systems. The configuration of each system is displayed in Table 2. Figure 2 illustrates the layered architecture of the experimental configuration. The lowest layer indicates the infrastructure which contains desktop computers in our experiments. The middle layer is where the Aneka platform runs. The top layer displays our application which is an interface to handle the execution of a given workflow application. We have used '.Net framework 4.5' to interact with Aneka libraries to send and monitor the execution of workflow tasks in the containers of the configured Aneka platform. Aneka supports task models to run specific commands besides automatically transferring I/O files of the tasks. However, Aneka does not support the workflow task model. Therefore, a workflow coordinator is developed to parse an XML DAG file and submit ready tasks to the Aneka task scheduler. Budget and deadlines are defined as QoS parameters for Aneka. Our workflow coordinator, after parsing XML, creates a list of tasks. Afterward, Task leveling, the DeadlineDistribution, and BudgetDistribution algorithms are invoked. Finally, the task distribution algorithm sends tasks for the task scheduler algorithm to the Aneka PaaS. These steps are developed in our workflow coordinator and are shown in Fig. 2. The proposed instance provisioning algorithm is developed as a custom scheduler algorithm in the Aneka Platform that runs on the master container. Aneka has two types of containers: Master and Worker containers. The Master container is responsible for scheduling tasks and monitoring their

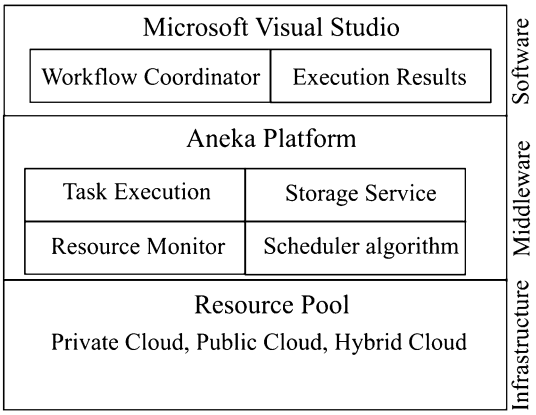


Fig. 2 Layered architecture of empirical experiment setup

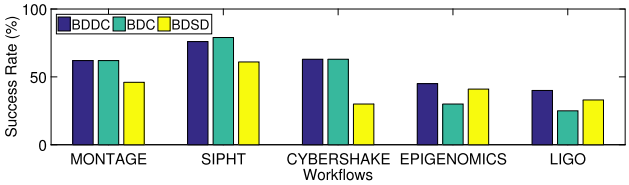


Fig. 3 Average success rates in empirical experiments

execution. Worker containers receive tasks from the master container and run them.

7.1.2 Empirical execution results

Figure 3 displays the results of empirical experiments with described scenarios in Sect. 7. Due to the long execution and large file transferring of scientific workflows and our limited resources; we have used a small number of tasks, that is, Montage with 100 tasks, Sipht with 30 tasks, Ligo with 30 tasks, Cybershake with 30 tasks, and Epigenomics with 24 tasks. Figure 4 displays DAGs of these five scientific workflows.

7.2 Experimental results

Experimental results for time ratio and cost ratio are depicted by box and whisker plot. Five summaries of statistics are legible: min, max, median, and first and third quartile. The values less or equal to 1.0 ($<=1$) for time and cost ratio imply that the constrained value of time or cost is satisfied. Lower values indicate higher QoS of the acquired cost or time. In our experiments, each workflow is tested with 100 task numbers. We observed that experiments by different task numbers (i.e., 50, 100, and 1000) of scientific workflows have very similar comparison results [8].

7.2.1 Montage Workflow

Figure 5 depicts the experimental results of the Montage workflow. Figure 5a shows that BDDC acquires the lowest cost ratio among all algorithms by satisfying the deadline constraint.

Figure 5b and d illustrate that BDC and BDDC have equal success rates and outperformed other algorithms in the various deadline and budget constraints.

Due to the page limitation, we have not depicted cost ratios under budget constraints and time ratios under deadline constraints. However, they can be almost implied from the combination of success rate, time ratio, and cost ratio. As can be seen, DBCS acquires the lowest cost ratio in the strict deadlines; it however could not earn the same performance in the loose deadline constraints. DBCS algorithm provides cheaper cost with the expensive of slower times. Therefore, in Fig. 5c when the budget constraint is consistent with 0.1 and the deadline range is variant between 0.1 to 0.6, DBCS provides almost the satisfying time ratio. Its success rate is zero for the mentioned constraints because it exceeds its cost ratio in the tight budget constraint of 0.1 (which is implied from Fig. 5c and d). In other words, DBCS loses the balance of time and cost; therefore, it acquires a valid time ratio when the budget is consistent to 0.1 in Fig. 5c but not valid cost ratios as we can imply from Fig. 5d; this led to zero success rate.

To understand the experiment results in Fig. 5, we must first analyze the Montage workflow characteristic. As shown in Fig. 4, each row has one type of task. Additionally, the second row of the Montage owns the maximum number of tasks. Montage workflow is an I/O-intensive workflow [19]. Maximum data communication and computation of the workflow are in the first and second row tasks. For example, in a Montage workflow with 100

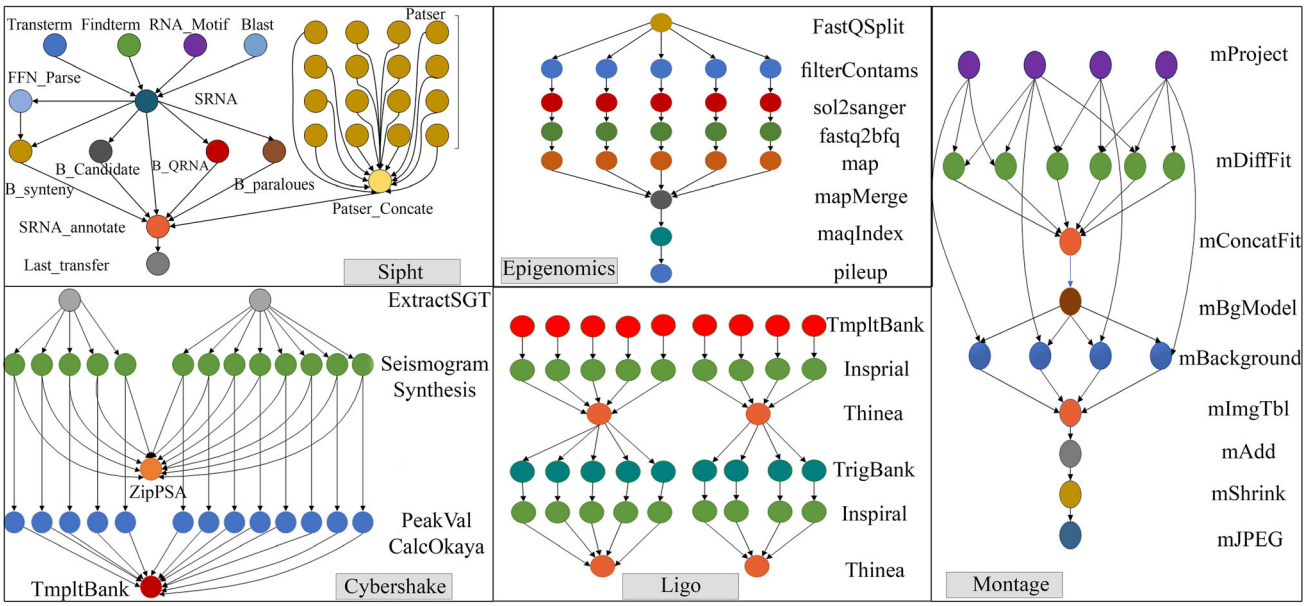
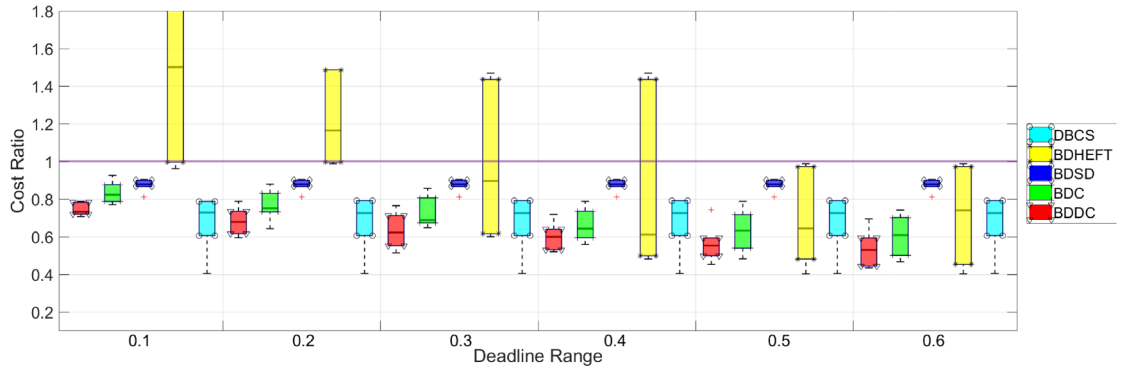
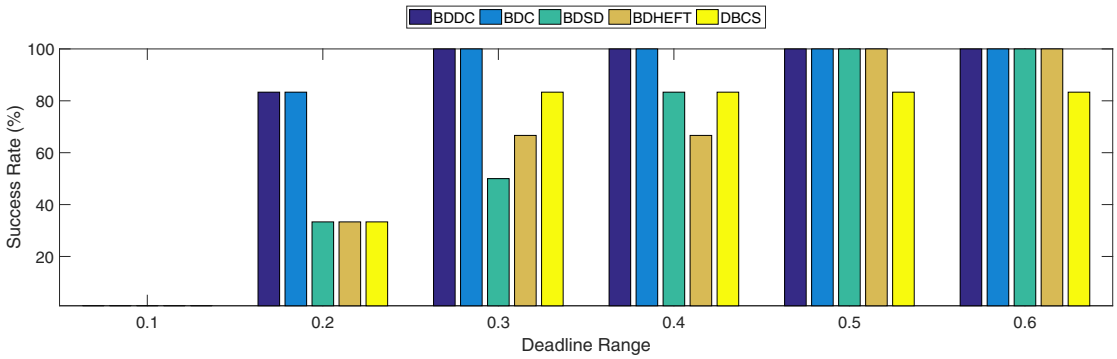


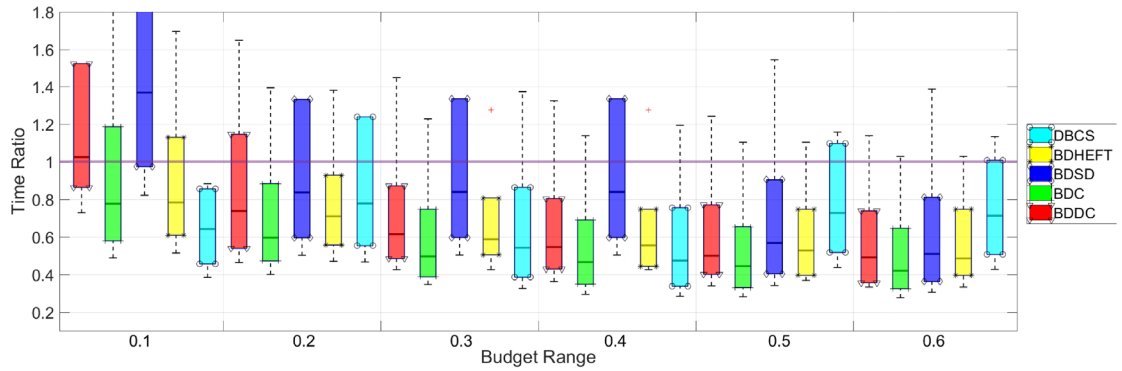
Fig. 4 Scientific workflows



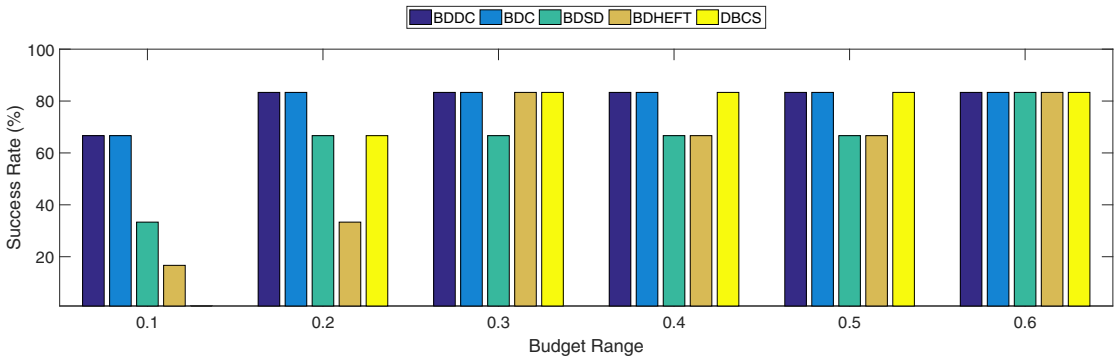
(a) Cost ratio under different deadline constraints



(b) Success rate under different deadline constraints



(c) Time ratio under different budget constraints



(d) Success rate under different budget constraints

Fig. 5 Montage workflow

tasks, 16 tasks belong to the first row, and 62 tasks belong to the second row, consisting of the workflow's biggest computation and communication time. Data communication is high between the tasks of the first and the second rows. A successful scheduling strategy would be provisioning instances with the maximum number of first-row tasks (reduce computation time using parallelism) and schedule the second-row tasks to the instances that contain their precedent tasks (reduce communication time). The method presented in Sect. 5.5 would help to reach the mentioned strategy. Meanwhile, Critical path based algorithms need more instances considering the many tasks in the second row of the Montage workflow.

7.2.2 Epigenomics workflow

Figure 6 contains the results of the Epigenomics workflow. Figure 6a, b depict the achieved experimental results for cost ratio and success rate under different deadlines. Figure 6a and b show that BDDC achieves the lowest cost under strict deadlines by earning the highest success rate among the other algorithms. BDC and BDHEFT achieve the same success rate in various deadline constraints. Meanwhile, Fig. 6a and c exhibit that BDC achieves a lower cost ratio and time ratio than the BDHEFT algorithm. Figure 6c shows the time ratio of algorithms over different budget constraints. It exhibits that BDSD accomplishes a higher success rate and lower time ratio under very strict budget constraints (like 0.1 and 0.2); however, Fig. 6d proves that it exceeds its demanded budget constraint in the strict budget constraints. Indeed, BDDC under the 0.3 budget constraint, gains the lowest time ratio and highest success rate. An Epigenomics workflow, which is a computation-intensive workflow [19], is depicted in Fig. 4. The fifth row of the Epigenomics workflow consists of MAP type tasks in which the workflow's main runtime belongs to this type of task [19]. BDC is not too conservative for budget; therefore, in the first four rows of the Epigenomics workflow tasks, it consumes enough budget that it can not continue without exceeding its demanded budget for the strict budget constraints. Only the resources with less monetary cost to the proportion of the remaining budget to the number of remaining tasks could be candidates for executing the task in the BDSD algorithm. This method would be effective when runtimes of the workflow tasks are too variant; therefore, it is not too efficient in the Epigenomics workflow. BDDC is a conservative algorithm for both budget and deadline. Therefore, it saves enough money to execute the first four rows of the workflow tasks that the rest of the budget could accommodate the provisioning costs for the fifth row. Hence, it outperforms other algorithms in the Epigenomics workflow. DBCS algorithm tends to be a conservative budget algorithm; therefore, it saves budget in the first four rows of the Epigenomics algorithm (the part with the lowest runtime). However, it can not found a balance between time ratio and cost ratio, but it has a better performance than the rest of the algorithms.

7.2.3 Sipht workflow

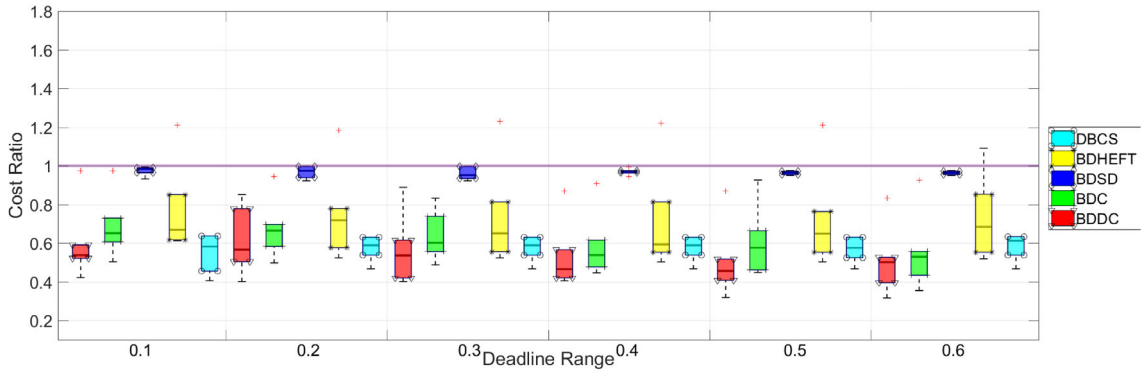
The results of the Sipht workflow are depicted in Fig. 7. Cost ratios and success rates under various deadline constraints are displayed in Fig. 7a and b. Figure 7a shows that the BDDC algorithm outperforms other algorithms by achieving a lower cost ratio in different deadline constraints. Figure 7b illustrates that BDDC, BDC, and BDHEFT success rates in strict deadlines (i.e., 0.1 and 0.2) are almost equivalent. Nevertheless, Fig. 7d shows that BDDC and BDSD in the strict budget constraint 0.1 have the highest success rate, and BDC and BDHEFT have the highest success rate for deadline constraint 0.2. Figure 7c shows that BDHEFT provides the fastest schedule plan. BDDC in the most strict budget constraint (0.1) accompanied by the BDSD algorithm, outperform other algorithms. Though, BDDC performance surpasses the BDSD algorithm in the other constraints for the Sipht workflow.

7.2.4 Cybershake workflow

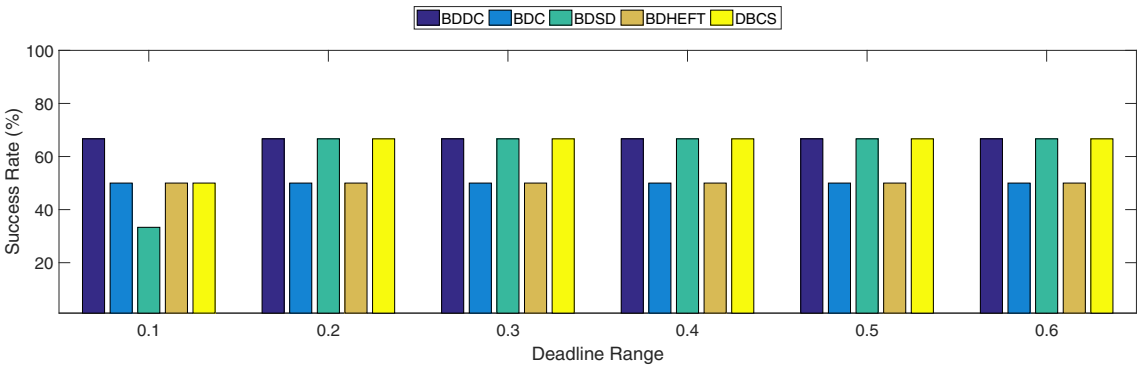
Figure 8 displays the experimental results of the utilizing success rate in the various deadline and budget constraints. Figure 8a exhibits that BDDC and BDC outperform other algorithms in strict deadlines. Besides, BDDC in the most strict deadline (0.1) outperforms the BDC algorithm; however, for the strict deadline of 0.2, it is the other way around. Figure 8a and b show that BDHEFT earns a lower success rate in the various deadline and budget constraints. BDHEFT cost exceeds its demanded budget, meaning that it sacrifices cost for achieving a faster schedule plan.

7.2.5 Ligo workflow

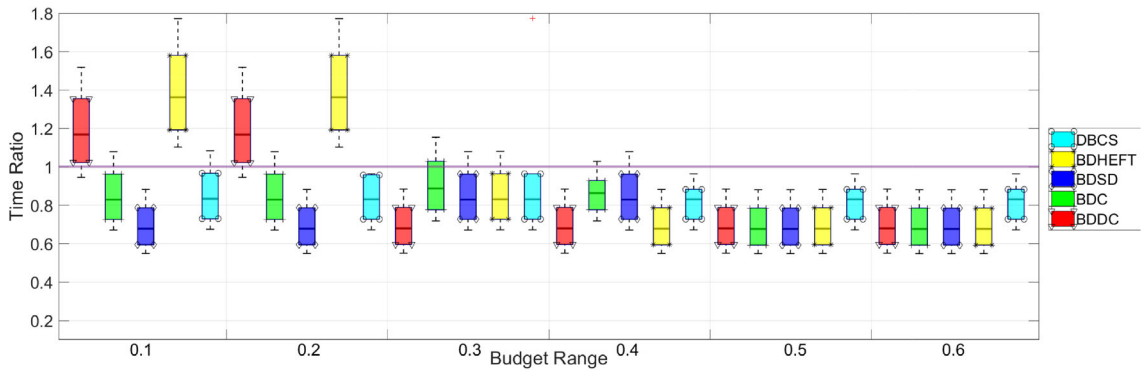
Figure 9 depicts the Ligo workflow's experimental results through the success rate in the various deadline and budget constraints. Figure 9a proves that the BDC algorithm outperforms all algorithms in strict deadlines. Besides, only BDC and BDSD can acquire a 100% success rate in the loose deadline constraint of 0.6. Figure 9b exhibits that BDC in strict budget constraints outperforms other algorithms by acquiring a higher success rate. Additionally, BDDC is the second algorithm to obtain better overall performance for earning higher success rates.



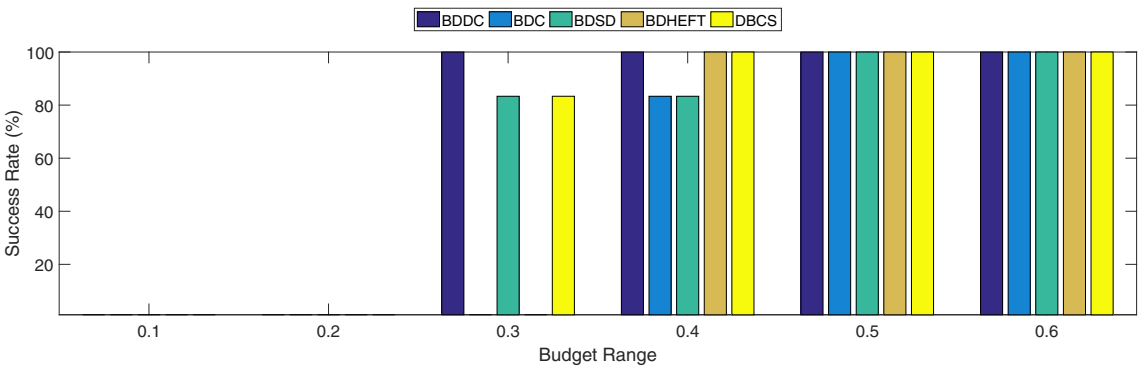
(a) Cost ratio under different deadline constraints



(b) Success rate under different deadline constraints

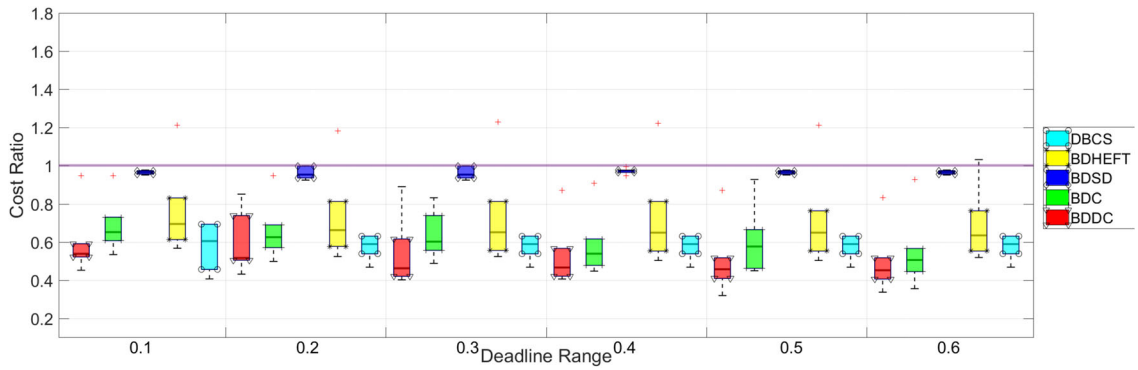


(c) Time ratio under different budget constraints

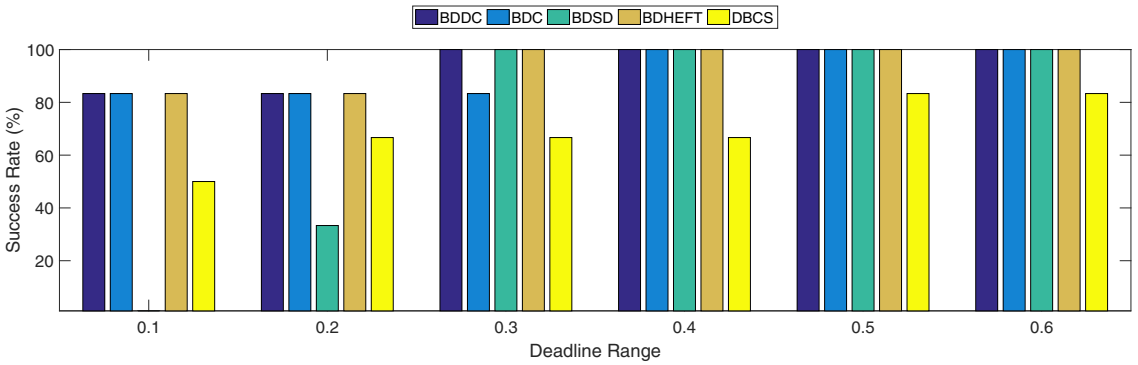


(d) Success rate under different budget constraints

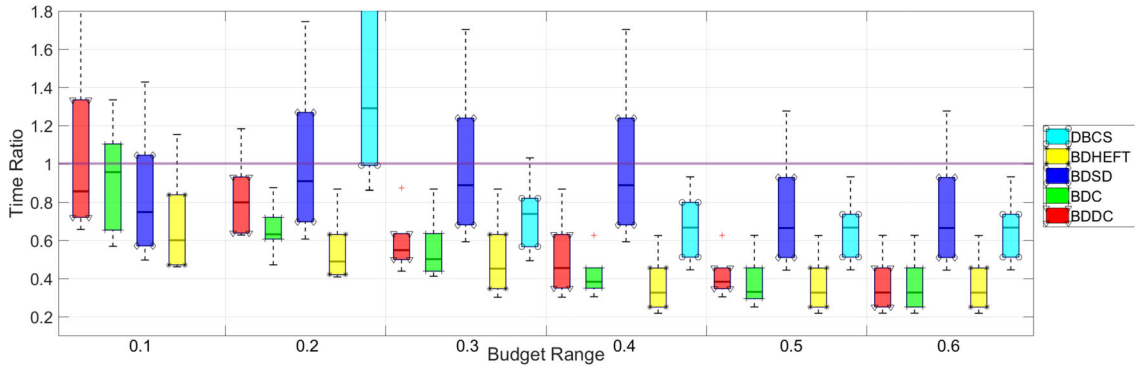
Fig. 6 Epigenomics workflow



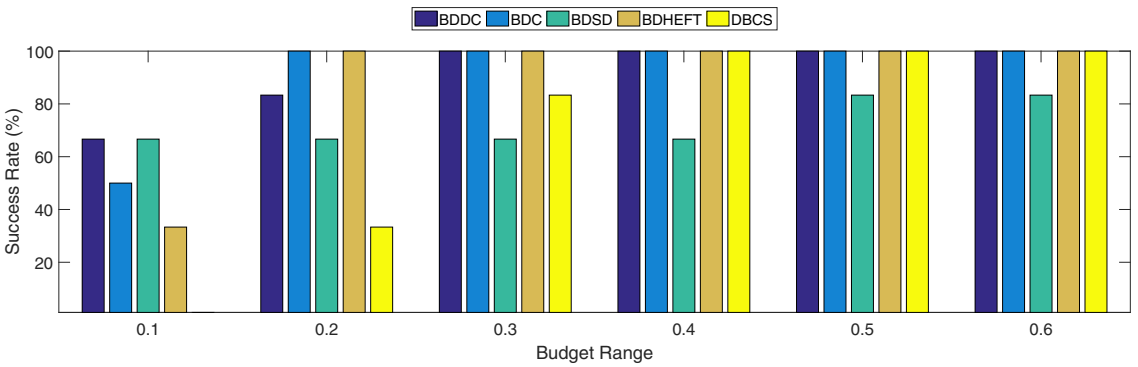
(a) Cost ratio under different deadline constraints



(b) Success rate under different deadline constraints

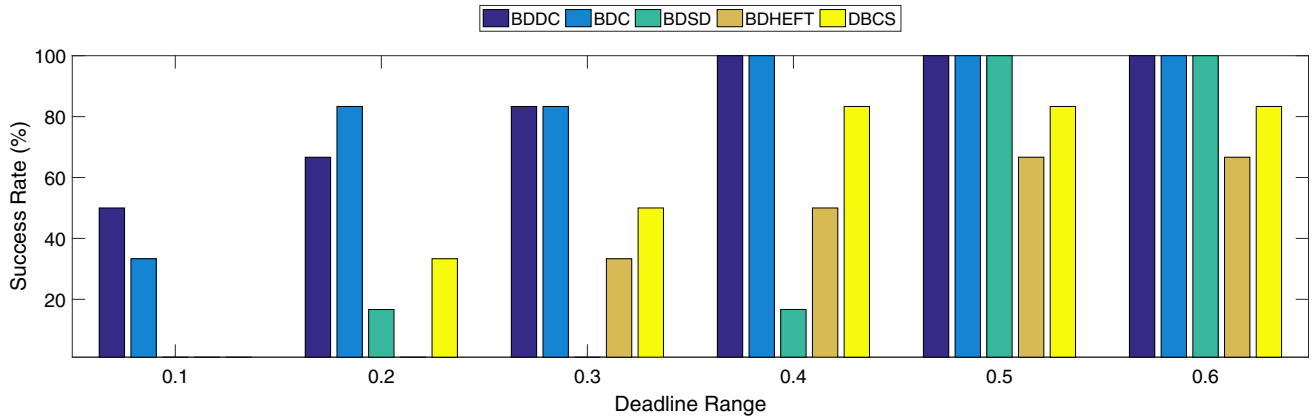


(c) Time ratio under different budget constraints

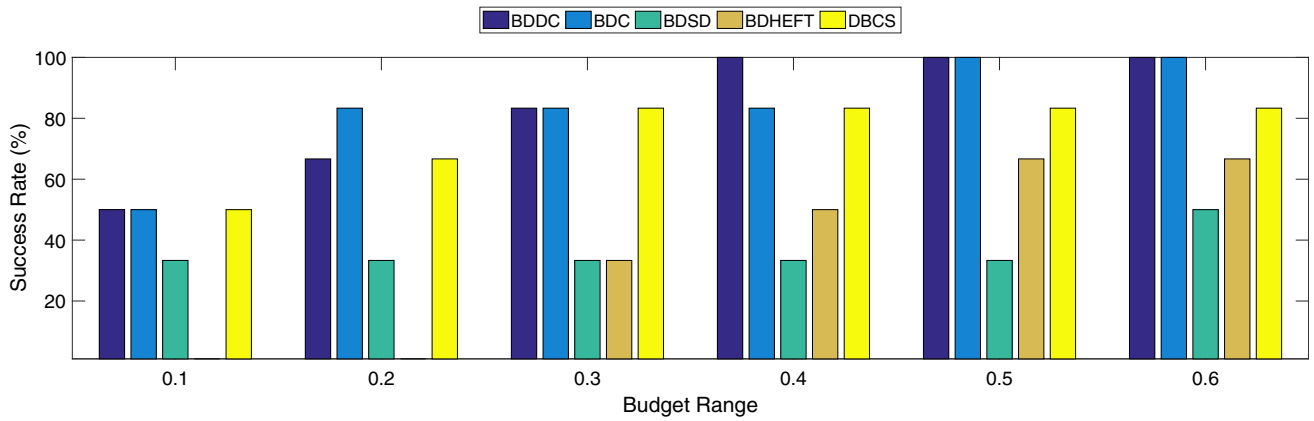


(d) Success rate under different budget constraints

Fig. 7 Sipht workflow



(a) Success rate under different deadline constraints



(b) Success rate under different budget constraints

Fig. 8 Cybershake workflow

7.2.6 Utilization rate

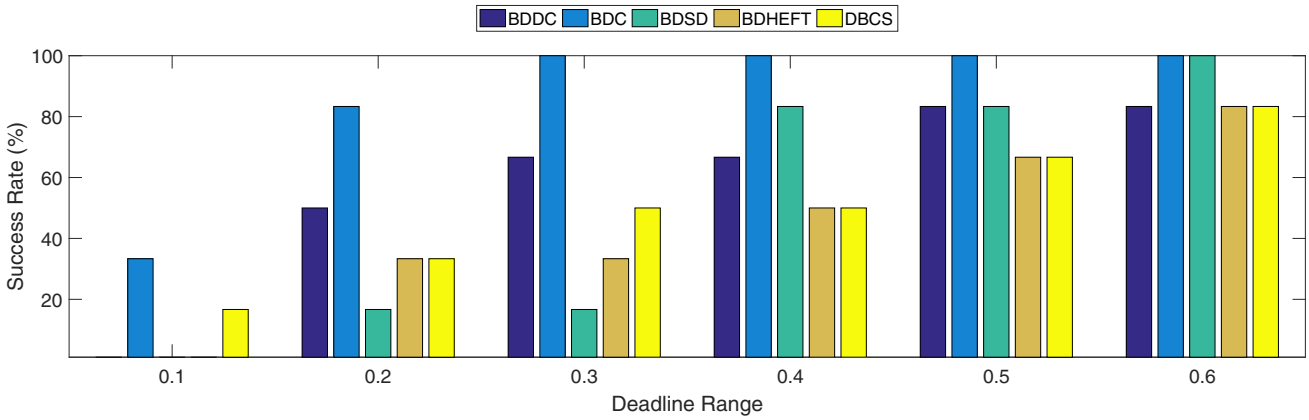
One of the key parameters for scheduling is the utilization rate of resources. A higher utilization rate usually implies better performance and thus typically leads to lower costs in the long run. Figure 10 shows the mean utilization rate of algorithms for different scientific workflows. It illustrates that the BDDC algorithm provides a more utilized schedule plan, as compared with other algorithms, for most workflows; and thus it deems to less economic costs. Figure 10 shows that BDC earns a lower mean utilization rate in some workflows, and the experimental results prove that BDC prefers to earn a faster schedule time than the economic cost. However, BDC was able to provide upstanding success rates. BDDC outperforms other algorithms in the utilization rate in Epigenomics, Sipht, and Ligo workflows. Moreover, DBCS is the second algorithm that achieves a higher utilization rate for Sipht and Cybershake workflows.

7.3 Analysis of ω (weight) impact on the success rate

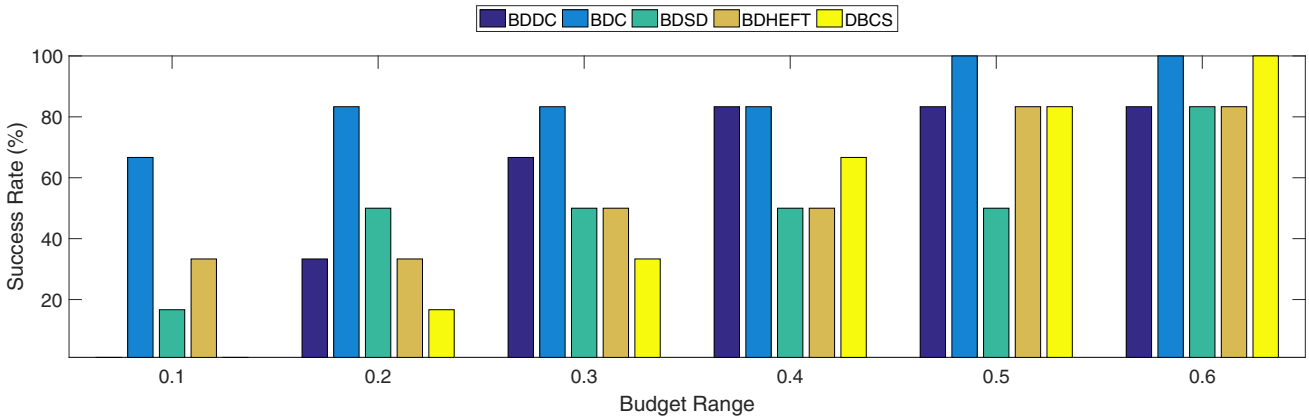
Based on the user priority for deadline and budget, the value of ω is utilized in Eq. (15). When ω is close to 1, it indicates that the deadline is more important than the budget. Figure 11 demonstrates the success rates of meeting the user constraints with different values initiated for ω by Eq. (15). The results indicate that the increasing impact of cost to time leads to higher success.

8 Conclusion

In this paper, we proposed two algorithms addressing the issues of budget-deadline constrained workflow scheduling for cloud IaaS. The presented algorithms are evaluated under various constraints to compare their cost and time efficacy, success rate, and utilization rate. The results are verified through empirical experiments. The experiments indicate that the BDDC algorithm obtains economic costs



(a) Success rate under different deadline constraints



(b) Success rate under different budget constraints

Fig. 9 Ligo workflow

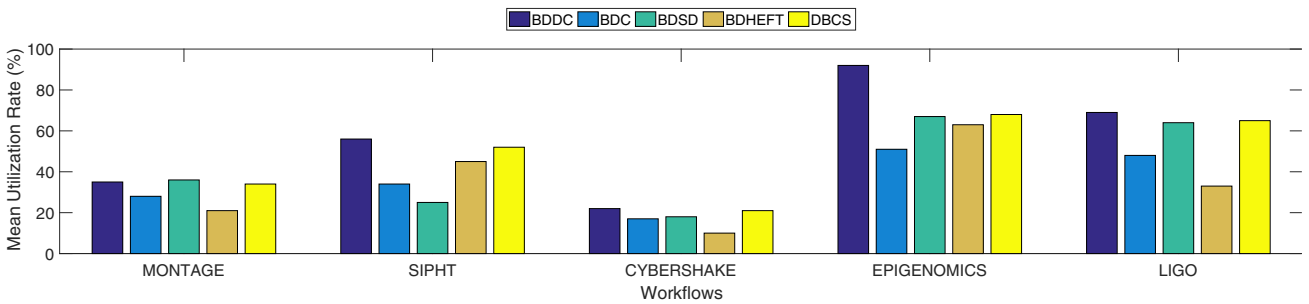


Fig. 10 Utilization rate for different workflows

while earning a higher success rate. The BDC algorithm acquires a higher success rate while earning the fastest schedule plan. Both BDDC and BDC outperform other algorithms in strict constraints for various evaluation metrics. Furthermore, the overhead of instance provisioning is more sensible in the real cloud environment. Our

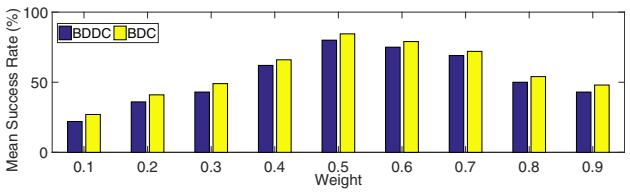


Fig. 11 Mean success rate of the proposed algorithms in different ω(weight)

experimental results also exhibit that the BDDC algorithm outperforms other algorithms in the mean utilization rate, and this in turn could result in more economic schedules. BDDC and BDC are suitable for budget-deadline constrained environments. In general, BDDC and BDC outperform other algorithms by providing a higher success rate and a higher QoS and utilization rate. For future work, we intend to investigate BDDC and BDC algorithms' performance in the dynamic scheduling plan with multiple types of workloads.

References

1. Abazari, F., Analoui, M., Takabi, H., Fu, S.: Simulation modelling practice and theory MOWS: multi-objective workflow scheduling in cloud computing based on heuristic algorithm. Simul. Modell. Pract. Theory **93**, 119–132 (2019). https://doi.org/10.1016/j.simpat.2018.10.004
2. Abrishami, S., Naghibzadeh, M., Epema, D.H.J.: Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds. Future Gener. Comput. Syst. **29**(1), 158–169 (2013). https://doi.org/10.1016/j.future.2012.05.004
3. Ahmad, W., Alam, B., Ahuja, S., Malik, S.: A dynamic VM provisioning and de-provisioning based cost-efficient deadline-aware scheduling algorithm for Big Data workflow applications in a cloud environment. Clust. Comput. **24**(1), 249–278 (2021). https://doi.org/10.1007/s10586-020-03100-7
4. Almi, K., Lee, Y.C., Mans, B.: On efficient resource use for scientific workflows in clouds. Comput. Netw. **146**, 232–242 (2018). https://doi.org/10.1016/j.comnet.2018.10.003
5. Aneka (2021). http://manjrasoft.com/
6. Arabnejad, H., Barbosa, J.G., Prodan, R.: Low-time complexity budget-deadline constrained workflow scheduling on heterogeneous resources. Future Gener. Comput. Syst. **55**, 29–40 (2016). https://doi.org/10.1016/j.future.2015.07.021
7. Arabnejad, V., Bubendorfer, K., Ng, B.: Budget distribution strategies for scientific workflow scheduling in commercial clouds. In: IEEE 12th International Conference on e-Science Budget, pp. 137–146. IEEE (2016)
8. Arabnejad, V., Bubendorfer, K., Ng, B.: Budget and deadline aware e-science workflow scheduling in clouds. IEEE Trans. Parallel Distrib. Syst. **30**(1), 29–44 (2019). https://doi.org/10.1109/TPDS.2018.2849396
9. Arabnejad, V., Bubendorfer, K., Ng, B.: Dynamic multi-workflow scheduling: a deadline and cost-aware approach for commercial clouds. Future Gener. Comput. Syst. **100**, 98–108 (2019). https://doi.org/10.1016/j.future.2019.04.029
10. Begnum, K.: Simplified cloud-oriented virtual machine management with MLN. J. Supercomput. **61**(2), 251–266 (2012). https://doi.org/10.1007/s11227-010-0424-0
11. Cadorel, E., Coullon, H., Menaud, J.m., Cadorel, E., Coullon, H., A, J.m.M., Cadorel, E., Atlantique, I.M.T.: A workflow scheduling deadline-based heuristic for energy optimization in Cloud. In: 15th IEEE International Conference on Green Computing and Communications, pp. 1–10. IEEE, Atlanta, United States (2019)
12. Casas, I., Taheri, J., Ranjan, R., Wang, L., Zomaya, A.Y.: GA-ETI: an enhanced genetic algorithm for the scheduling of scientific workflows in cloud environments. J. Comput. Sci. **26**, 318–331 (2018). https://doi.org/10.1016/j.jocs.2016.08.007
13. Casas, I., Taheri, J., Ranjan, R., Zomaya, A.Y.: PSO-DS: a scheduling engine for scientific workflow managers. J. Supercomput. **73**(9), 3924–3947 (2017)
14. Chakravarthi, K., Shyamala, L., Vaidehi, V.: Budget aware scheduling algorithm for workflow applications in IaaS clouds. Clust. Comput. **4**, 3405–3419 (2020). https://doi.org/10.1007/s10586-020-03095-1
15. Chang, Y.-W., Hsu, P.Y.: An empirical investigation of organizations ' switching intention to cloud enterprise resource planning : a cost-benefit perspective. Inf. Dev. **35**(2), 290–302 (2019). https://doi.org/10.1177/0266666917743287
16. Chirkin, A.M., Belloum, A.S., Kovalchuk, S.V., Makkes, M.X., Melnik, M.A., Visheratin, A.A., Nasonov, D.A.: Execution time estimation for workflow scheduling. Future Gener. Comput. Syst. **75**, 376–387 (2017). https://doi.org/10.1016/J.FUTURE.2017.01.011
17. Deelman, E., Vahi, K., Juve, G., Rynge, M., Callaghan, S., Maechling, P.J., Mayani, R., Chen, W., da Silva Ferreira, R., Livny, M., Wenger, K.: Pegasus, a workflow management system for science automation. Future Gener. Comput. Syst. **46**, 17–35 (2015). https://doi.org/10.1016/j.future.2014.10.008
18. Ghafouri, R., Movaghar, A., Mohsenzadeh, M.: A budget constrained scheduling algorithm for executing workflow application in infrastructure as a service clouds. Wirel. Pers. Commun. **103**(3), 2035–2070 (2018). https://doi.org/10.1007/s11277-018-5895-y
19. Juve, G., Chervenak, A., Deelman, E., Bharathi, S., Mehta, G., Vahi, K.: Characterizing and profiling scientific workflows. Future Gener. Comput. Syst. **29**, 682–692 (2013). https://doi.org/10.1016/j.future.2012.08.015
20. Kim, J., Lee, K.: I/O resource isolation of public cloud serverless function runtimes for data-intensive applications. Clust. Comput. **23**(3), 2249–2259 (2020). https://doi.org/10.1007/s10586-020-03103-4
21. Kim, S., Suh, Y.K., Kim, J.: EXTES: an execution-t ime estimation scheme for efficient computational science and engineering simulation via machine learning. IEEE Access **7**, 98993–99002 (2019). https://doi.org/10.1109/ACCESS.2019.2929800
22. Managing your costs with AWS Budgets—AWS Billing and Cost Management. https://docs.aws.amazon.com
23. Pandey, S., Karunamoorthy, D., Buyya, R.: Workflow engine for clouds. Cloud Comput. (2011). https://doi.org/10.1002/9780470940105.ch12
24. Suh, Y.K., Kim, S., Kim, J.: CLUTCH: a clustering-driven runtime estimation scheme for scientific simulations. IEEE Access **8**, 220710–220722 (2020). https://doi.org/10.1109/ACCESS.2020.3042596
25. Sun, T., Xiao, C., Xu, X.: A scheduling algorithm using sub-deadline for workflow applications under budget and deadline constrained. Clust. Comput. **22**(3), 5987–5996 (2019). https://doi.org/10.1007/s10586-018-1751-9

26. Topcuoglu, H., Hariri, S., Wu, M.Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**(3), 260–274 (2002). https://doi.org/10.1109/71.993206
27. Verma, A., Kaushal, S.: Cost-time efficient scheduling plan for executing workflows in the cloud. J. Grid Comput. **13**(4), 495–506 (2015). https://doi.org/10.1007/s10723-015-9344-9
28. Xie, G., Zeng, G., Xiao, X., Li, R., Li, K.: Energy-efficient scheduling optimization for parallel applications on heterogeneous distributed systems. IEEE Trans. Parallel Distrib. Syst. **28**(12), 3426–3442 (2017). https://doi.org/10.1142/S0218126620502035
29. Yuan, Y., Li, X., Wang, Q., Zhang, Y.: Bottom level based heuristic for workflow scheduling in grids. Chin. J. Comput. **31**(2), 282 (2008)
30. Zheng, W., Qin, Y., Bugingo, E., Zhang, D., Chen, J.: Cost optimization for deadline-aware scheduling of big-data processing jobs on clouds. Future Gener. Comput. Syst. **82**, 244–255 (2018). https://doi.org/10.1016/j.future.2017.12.004
31. Zheng, W., Sakellariou, R.: Budget-deadline constrained workflow planning for admission control. J. Grid Comput. **11**(4), 633–651 (2013). https://doi.org/10.1007/s10723-013-9257-4
32. Zhou, N., Lin, W., Feng, W., Shi, F., Pang, X.: Budget-deadline constrained approach for scientific workflows scheduling in a cloud environment. Clust. Comput. (2020). https://doi.org/10.1007/s10586-020-03176-1
33. Zhu, Z., Tang, X.: Deadline-constrained workflow scheduling in IaaS clouds with multi-resource packing. Future Gener. Comput. Syst. **101**(December), 880–893 (2019). https://doi.org/10.1016/j.future.2019.07.043

Ahmad Taghinezhad-Niar was born in Ardabil, Iran in 1993. He is currently a Ph.D. Candidate in Computer Engineering at the University of Tabriz, Iran. He received his M.S. degree and B.S. in computer engineering from University of Tabriz, Iran, and Bahonar University of Shiraz, Iran in 2017 and 2015, respectively. His research interests include distributed systems and cloud computing. He is a lecturer at University of Tabriz and University of Mohaghegh Ardabili, Ardabil, Iran since 2017.

Saeid Pashazadeh received the B.Sc. degree in computer engineering from the Sharif University of Technology, Tehran, Iran, in 1995, and the M.Sc. and Ph.D. degrees in computer engineering from the Iran University of Science and Technology, Tehran, in 1998 and 2010, respectively. He is currently an Associate Professor with the Department of Information Technology, Faculty of Electrical and Computer Engineering, University of Tabriz, Tabriz, Iran. His research interests include wireless sensor networks, target tracking, formal methods, distributed systems, and stochastic systems.

Javid Taheri is a Full Professor at the Department of Computer Science at Karlstad University, Sweden. He received his Ph.D. in Mobile Computing from University of Sydney (Australia) in 2007, and his Bachelor and Masters of Electrical Engineering from Sharif University of Technology, Tehran (Iran) in 1998 and 2000, respectively. He is the recipient of many awards including being selected as one of the top 200 young researchers in the world by the Heidelberg Forum in 2013, the recipient of several best paper awards since 2007, and the recipient of the prestigious IEEE Middle Career Researcher award from TSCS in Scalable Computing in 2019. He also holds several cloud/networking related industrial certification from VMware, Cisco, Microsoft and IBM. His research interests includes Cloud Computing, Edge/Fog Computing, Network Function Virtualization, Software-defined Networking, and AI-based optimization techniques. He is the editor of two books entitled "Big Data and Software Defined Networks", and "Edge Computing: Models, Technologies and Applications". He co-authored>170 scientific articles and papers, has been serving as an editor for>20 journals, as well as a member of organising team for>40 international conferences.