TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 10/22 pp112–126
DOI: 10.26599/TST.2022.9010065
Volume 29, Number 1, February 2024

Monte Carlo Simulation-Based Robust Workflow Scheduling for Spot Instances in Cloud Environments

Quanwang Wu, Jianzhao Fang, Jie Zeng*, Junhao Wen, and Fengji Luo

Abstract: When deploying workflows in cloud environments, the use of Spot Instances (SIs) is intriguing as they are much cheaper than on-demand ones. However, SIs are volatile and may be revoked at any time, which results in a more challenging scheduling problem involving execution interruption and hence hinders the successful handling of conventional cloud workflow scheduling techniques. Although some scheduling methods for SIs have been proposed, most of them are no more applicable to the latest SIs, as they have evolved by eliminating bidding and simplifying the pricing model. This study focuses on how to minimize the execution cost with a deadline constraint when deploying a workflow on volatile SIs in cloud environments. Based on Monte Carlo simulation and list scheduling, a stochastic scheduling method called MCLS is devised to optimize a utility function introduced for this problem. With the Monte Carlo simulation framework, MCLS employs sampled task execution time to build solutions via deadline distribution and list scheduling, and then returns the most robust solution from all the candidates with a specific evaluation mechanism and selection criteria. Experimental results show that the performance of MCLS is more competitive compared with traditional algorithms.

Key words: constrained optimization; Monte Carlo simulation; robustness; Spot Instances (SIs); workflow scheduling

1 Introduction

Cloud computing has already been a mainstream paradigm for providing computing resources in the industry. It provides a highly scalable pool of computation resources to execute real-world applications faster than ever. Moreover, it is of paramount importance for computation-intensive workflow applications, such as astronomy and genetic science, as it can substantially advance such applications and their corresponding domain sciences. Resorting to cloud computing, users can dynamically lease and end the use of computing resources to their needs and pay according to actual usage. Moreover, because there are usually diverse computing resources available, users can easily trade off the performance and execution costs of cloud resources by optimizing resource scheduling[1].

Generally, three pricing models are utilized for cloud resources, namely, on-demand, reserved, and spot, such as the ones in Amazon EC2. With On-demand Instances (OIs), users pay for the computation capacity by the hour, depending on the type of instances. Reserved instances provide a capacity reservation and some discounts based on a long-term commitment compared with on-demand pricing. Spot Instances (SIs) are introduced to encourage users to take advantage of the spare cloud capacity as volatile resources. From the perspective of users, the

• Quanwang Wu and Jianzhao Fang are with the College of Computer Science, Chongqing University, Chongqing 400044, China. E-mail: {wqw, fangjianzhao}@cqu.edu.cn.
• Jie Zeng is with the National Experimental Teaching Demonstration Center, Chongqing University, Chongqing 400044, China. E-mail: zj0101@cqu.edu.cn.
• Junhao Wen is with the College of Big Data and Software Engineering, Chongqing University, Chongqing 400044, China. E-mail: jhwen@cqu.edu.cn.
• Fengji Luo is with the School of Civil Engineering, The University of Sydney, Sydney 2006, Australia. E-mail: fengji.luo@sydney.edu.au.
∗ To whom correspondence should be addressed.

Manuscript received: 2022-11-02; revised: 2022-12-02; accepted: 2022-12-15

spot model is intriguing owing to its steeper discount compared with other models, but it is daunting for its instability. Specifically, SIs are much cheaper than OIs (e.g., the cost ratio between SIs and OIs can be as low as 10%), but they are volatile and may be revoked at any time[2, 3].

When SIs were first introduced at the end of 2009, they were run through an auction-like mechanism. The spot price was set by a provider, and it fluctuated periodically depending on the real-time demand and supply of cloud resources. Users needed to bid for spot resources by specifying the maximum price that they would pay. The bid is fulfilled when the bid price is higher than the current market price, and the SIs being used may be revoked any time when the current market price increases and exceeds the bid price. With the development of cloud computing, SIs have evolved by eliminating bidding and simplifying purchasing for the sake of easier usage, such as Amazon's latest EC2 SIs† and Google's spot virtual machines‡[4]. SI prices are set by providers and gradually adjusted based on long-term trends in the supply and demand for SI capacity. Consequently, the present prices of SIs are smooth and predictable. Nevertheless, SIs may still be interrupted and revoked when the provider needs them back for repurposing capacity, host maintenance, or other reasons. Hence, previously, the largest challenge in the use of SIs was how to bid appropriately, but at present, the obstacle is how to tackle the interruption, which is unpredictable.

SIs are quite promising for fault-tolerant and nonreal-time large scientific and business applications, such as big data analytics and media processing applications. These applications can be usually represented as a workflow through a direct acyclic graph, where a vertex stands for a task and an edge stands for a precedence dependency among tasks. When deploying workflows to cloud resources, the makespan of workflows and the total economic cost are two crucial user requirements, and several studies have been performed to optimize them[5]. However, these traditional workflow scheduling techniques are mainly targeted for OIs and cannot be directly applied to SIs because they are oblivious to resource interruption in runtime. Although some scheduling methods for SIs have been proposed[6], they are mainly targeted at the bidding strategies of SIs, which are no more applicable. Hence, an effective workflow scheduling approach for SIs is in an imperative demand.

† http://aws.amazon.com/ec2/spot/

‡ https://cloud.google.com/spot-vms

This study focuses on how to effectively deploy workflows on volatile SIs in cloud environments with awareness of usage costs and soft deadline constraint. A stochastic workflow scheduling model for SIs is developed, and a utility function is introduced to assess the user utility with regard to usage costs and deadline constraint satisfaction. Based on Monte Carlo simulation and list scheduling, we propose a stochastic scheduling method called MCLS to optimize the utility function. According to the Monte Carlo simulation framework, MCLS employs sampled task execution time to produce solutions via deadline distribution and list scheduling. It then puts promising solutions to the candidate pool and returns the best one as the output depending on a specific solution evaluation mechanism and selection criteria. In the experiments, we compare MCLS with several state-of-the-art scheduling methods, and the experimental results demonstrate that the performance of MCLS is very competitive. In each setting, it achieves the lowest monetary cost and a relatively high success ratio for meeting deadlines.

The remainder of this paper is organized as follows. Section 2 presents a literature review. Section 3 describes the scheduling model and problem formulation. Section 4 proposes MCLS. Section 5 gives the experimental evaluations. Finally, Section 6 concludes this paper.

2 Related Work

In this section, we first present a review of representative works on workflow scheduling in stable and volatile clouds. Then, we review the literature on robust workflow scheduling in distributed systems.

2.1 Workflow scheduling in stable clouds

Stable cloud resources, such as OIs, are highly reliable in general. When deploying workflows in a stable cloud, there are generally three ways to compromise the cost and makespan: deadline-constrained scheduling, budget-constrained scheduling, and multiobjective scheduling[5, 7].

Deadline-constrained workflow scheduling aims to optimize the cost as much as possible under a deadline condition. Deadline distribution is a widely used method for this end. Based on the concept of the Partial Critical Path (PCP), Abrishami et al.[8] proposed two algorithms called IaaS Cloud PCP (IC-PCP) and IC-PCP with Deadline Distribution (IC-PCPD2) for scheduling workflows in the cloud. The former distributes the overall deadline to PCPs, whereas the latter further distributes the deadline to each task in proportion to its minimum execution time. Then, the cheapest resources can meet the latest completion time of the task for the partition, and task is selected. Arabnejad et al.[9] introduced a tunable cost-time trade-off for heterogeneous instances and proposed a novel approach that satisfies budget and deadline constraints. Experiments show that it achieves a 20% higher success rate than other algorithms. In Ref. [10], a width-changing trend-aware look-ahead workflow scheduling algorithm (namely W-LA) was proposed. Metaheuristics were also utilized to address deadline-constrained scheduling. For example, particle swarm optimization was used in Ref. [11], a particle's position was encoded to represent mappings between tasks and computation resources, and then a schedule generation method was designed to convert a particle's position into a schedule solution. The List and Ant Colony Optimization (L-ACO) method was proposed in Ref. [12] for deadline-constrained cost optimization, where an ant constructs an ordered task list according to the pheromone trail and local heuristic information, and then builds a solution based on it.

Budget-constrained scheduling aims to minimize the makespan under a budget constraint. For example, Faragardi et al.[13] ranked resources by their efficiency rate, and based on the classic list scheduling heuristic Heterogeneous Earliest Finish Time (HEFT)[14], they proposed the greedy resource provisioning-HEFT method to minimize the makespan while meeting the user-specified budget constraint. Ghafouri et al.[15] proposed a scheduling algorithm named constrained budget-decreased time, where the scheduling of critical and noncritical tasks is combined with a backtracking heuristic to obtain a small completion time. A fair budget-constrained workflow scheduling algorithm was proposed in Ref. [16], where a cost-saving mechanism was introduced to save the budget by adjusting the cost-time efficient factor.

Multiobjective scheduling aims to optimize the makespan and economic cost by producing a set of tradeoff optimal solutions. It has received increasing research attention these years. In Ref. [17], Durillo and Prodan extended the HEFT for multiobjective optimization, and a set of nondominated partial solutions was kept in each task allocation step. In Ref. [18], a multiobjective evolutionary list scheduling was proposed for minimizing costs and makespan, where a list scheduling heuristic and an evolutionary algorithm were seamlessly combined to have complementary advantages. In Ref. [19], Zhou et al. proposed a new list scheduling algorithm to solve the multiobjective workflow scheduling problem, where fuzzy dominance was applied to measure the relative fitness of solutions in a multiobjective solution space.

2.2 Workflow scheduling in volatile clouds

Aside from stable cloud resources, cloud providers also provide their spare resources to users in a volatile form at a low price (i.e., SIs). Cheap but volatile SIs can be revoked by providers at any time. Up to present, much research efforts have been devoted to SIs due to their extraordinary nature. For instance, the pricing mechanism of a spot market in cloud computing was investigated in Ref. [20], and the Amazon EC2 SI pricing was deconstructed in Ref. [21]. Researchers in Refs. [2, 22, 23] focused on how to predict the spot price and bid for SIs, and those in Ref. [24] studied how to make use of checkpoints to alleviate the loss resulting from SI interruption. In Refs. [25–27], researchers investigated how to use SIs for Internet-based services, machine learning, and SARS-CoV-2 DNA sequence comparison, respectively.

Some research efforts have also examined running workflows on SIs. Xu et al.[28] proposed a cost-effective transient server provisioning framework called iSpot to achieve a predictable performance in the cloud. It adopts a price prediction method based on long short-term memory for automatic job profiling to improve the reliability of workflow execution. A deadline-constrained workflow scheduling method was proposed in Ref. [29]. It uses OIs to perform critical path tasks and SIs to perform highly concurrent fine-grained tasks. When a workflow is interrupted in runtime, some SIs used to excute it may be replaced by OIs. Martinez et al.[30] constructed a Markov decision process for the workflow execution and search for the optimal policy by considering user preferences in time and cost. The optimal solution is produced offline before execution, and actions selected on-the-fly depend on the occurrence of instance revocations and the actual task completion time. Ghavamipoor et al.[31] employed an artificial neural network to define a failure prediction module for SIs, and proposed a reliability-aware scheduling algorithm for minimizing the makespan of workflows considering a minimum ensured reliability rate. Some studies combined multiple strategies to address the volatility when executing workflows in SIs. For example, in Ref. [6], task duplication and resubmission techniques were adopted to mitigate the side effect resulting from SI failures.

The above studies on workflow scheduling are targeted at the out-of-date bidding-based SIs, and they mainly focus on how to predict the performance and how to bid, even though some papers were published after bidding-based SIs have become obsolete. In these days, SIs have evolved by eliminating bidding and simplifying purchasing for the sake of easy usage, and their prices are smooth and predictable[32].

Some recent studies have been conducted for the latest cloud spot market. Cost-minimizing reservation and SI scheduling were studied in Ref. [4], and two algorithms were proposed with and without the assumption of known future demands. An optimization framework FarSpot for workflow applications was proposed in Ref. [33] to minimize the cost within the performance constraints. By assigning a sub-deadline to each task, FarSpot migrates tasks dynamically among SIs to reduce the cost based on the assumption that the live migration function is always available for SIs, which usually does not hold in reality because of business reasons. Teylo et al.[34] presented a hibernation-aware dynamic scheduler, which schedules bag-of-task applications with deadline constraints in SIs and OIs for minimizing monetary costs. It employs a dynamic scheduling module that applies task migration and work-stealing techniques to address the issue of SI failures. Pham and Fahringer[35] considered the fulfillment and interruption rates of volatile resources to model the instability of SIs, and employed an evolutionary algorithm to generate a set of tradeoff solutions.

2.3 Robust workflow scheduling

Robust scheduling (also known as proactive scheduling) aims to develop a schedule that can cope with uncertainties during execution. To enhance the robustness of executing workflows in a volatile distributed system, the main techniques include replication, checkpointing, and statistics[5]. Moreover, reactive scheduling mechanisms, such as resubmission, are often employed as a supplement. Because robust workflow scheduling is not only limited to SIs, we reviewed it in a broader sense (i.e., distributed computing) below.

For replication, a task in the workflow may be replicated on multiple computation resources for execution more than once. Hence, it is also known as a redundancy-based scheduling. In Ref. [36], a two-step scheduling method was proposed for a heterogeneous distributed computing environment, in which a schedule was first produced to minimize the makespan, and then tasks, which are on the critical path, were replicated to enhance reliability. In Ref. [37], the concept of the weighted average makespan was introduced for improving the reliability of executing workflows in a multiprocessor system. Two scheduling algorithms were presented for fault-tolerant scheduling based on task replication and simulated annealing. In Ref. [6], how to enhance the reliability of workflow execution using task replication and SIs was investigated. In Ref. [38], a fault-tolerant scheduling algorithm was proposed for meeting the soft deadline of a workflow in cloud systems, where resubmission and replication are combined together to play their respective advantages for fault tolerance, while trying to meet the soft deadline. The side effect of this technology is that it inevitably leads to higher costs.

Checkpointing saves the states of a running task periodically to reliable storage, and with this technique, the task can be restarted from its last checkpoint or the saved state after a failure[24]. By employing checkpointing as a fault-tolerant strategy, a robustness metric called tolerance time was introduced in Ref. [39], which indicates the amount of time that a workflow can be delayed without violating the deadline constraint. Then, a robust scheduling algorithm was devised based on IC-PCP to optimize the cost and robustness. Zhou et al.[40] built a fault-tolerant framework combining the checkpointing mechanism and optimized the bidding strategy for SIs.

The statistics-based methods aim to produce a robust solution based on statistical information that absorbs some uncertainties during execution. In Ref. [41], Canon and Jeannot introduced a makespan distribution evaluation method for workflow applications, and then devised different strategies to optimize the makespan and robustness with an evolutionary metaheuristic. In Ref. [42], Zheng and Sakellariou proposed a Monte Carlo algorithm to schedule workflows in heterogeneous computing systems using random sampling to cope with unavoidable uncertainties in individual task execution times. In Ref. [43], a stochastic HEFT scheduling algorithm for grids was proposed, which incorporates the expected value and variance of the stochastic execution time into scheduling. In Ref. [44], Li et al. built a stochastic parallel application scheduling model for heterogeneous cluster systems, and proposed a Stochastic Dynamic Level Scheduling (SDLS) algorithm, which handles task time randomness based on stochastic scheduling attributes, such as stochastic bottom levels and stochastic dynamic levels.

This study designs a robust workflow scheduling method to optimize the cost within a deadline, for the latest SI market-this has not been resolved by the existing studies. To address the execution uncertainty, a Monte Carlo simulation framework is developed to yield a robust base schedule with OIs and adjust it through resubmission with SIs.

3 Scheduling Model

This section first describes the cloud workflow scheduling model, and then formulates the problem studied in this work.

3.1 Workflow and cloud resource model

A workflow application is usually represented by a directed acyclic graph, $\mathrm{DAG} = (V, E)$, where vertices V denote tasks in the workflow and directed edges E denote precedence dependencies between tasks. Each task $t_i \in V$ stands for an indivisible individual application with computation workload w_i. A precedence dependency $e_{i,j} \in E$ implies that task t_j can start only when task t_i finishes. The source and destination of $e_{i,j}$ are called the parent and child tasks (also known as predecessor and successor), respectively. A non-negative weight $d_{i,j}$ is associated with the edge $e_{i,j}$, which represents the amount of data that t_i sends to t_j. When $d_{i,j}$ is 0, t_j is executed once t_i finishes, and when it is larger than 0, t_j can be executed after the data from t_i have been received. Two dummy tasks t_{entry} and t_{exit} with no computation workload are set on the start and end of a workflow, respectively, and hence DAG is generalized with exactly one entry and one exit. A workflow sample is shown in Fig. 1, where the number in each task node indicates its computation workload, and the number on each edge indicates its data amount.

In cloud environments, a commercial cloud provider offers to its clients "infinite" virtualized computation

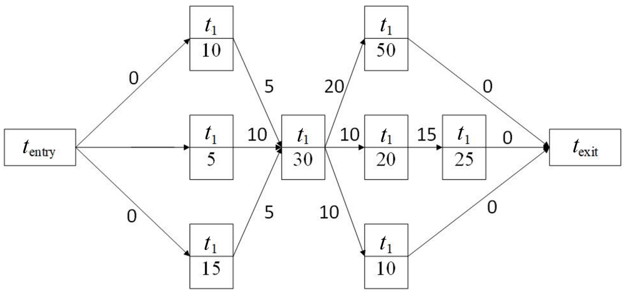

Fig. 1 Sample of workflow applications.

resources in different types, and each type specifies the processing capability and usage cost. As mentioned above, there are three pricing models for cloud resources in general: on-demand, reserved, and spot. In this work, we focus on how to take advantage of SIs to execute workflows for reducing expenses. Because SIs are volatile and may be revoked by providers at any time, we additionally employ OIs as a supplement for improving reliability. Hence, the cloud infrastructure we consider is a combination of SIs (volatile resources) and OIs (stable resources). Let $R = \{r_1, r_2, \ldots, r_{|R|}\}$ be a set of cloud resources used for executing workflows. For every r_l, we use s_l to represent its processing capability and p_l to represent the price of each billing interval Φ.

For an OI, users are charged depending on the number of time intervals that they have used cloud resources, and any partial utilization of an interval is rounded to a full one. SIs are volatile and may be revoked at any time when the provider needs them back because of repurposing capacity, host maintenance, or violating constraints. Such a reclaim is called an SI interruption, and from the perspective of spot users, they are generally agnostic on when SIs are revoked and how long a deployed SI will run in total before being interrupted[32, 45]. That is, the time point when an SI is interrupted from the lease start time cannot be predicted, and it can only be treated as a random variable. We denote this interruption time point as X_l, and when it is less than its lease end time, X_l is interrupted before the user finishes leasing it. When an SI is interrupted and revoked by the provider, the last partial utilization of a time interval is free of charge. Hence, given that r_l is leased to a user during the period from LST_l (lease start time) to LET_l (lease end time), then the usage cost can be calculated as

$$c_l = \begin{cases} \lceil (LET_l - LST_l)/\Phi \rceil \times p_l, \\ \qquad \text{if } r_l \text{ is OI or } (r_l \text{ is SI and } X_l > LET_l); \\ \lfloor (X_l - LST_l)/\Phi \rfloor \times p_l, \\ \qquad \text{if } r_l \text{ is SI and } X_l < LET_l \end{cases} \tag{1}$$

3.2 Problem formulation

When task t_i is allocated to an OI r_l, its execution time can be calculated as w_i/s_l. For an SI, the situation is more complicated as it may be interrupted during execution. When an SI is recycled, the supply of SIs may not be so sufficient at the moment. As the makespan is a crucial user requirement for workflows, it is better to restart the interrupted task on a stable OI for reliability instead of trying SIs again. Specifically, we stipulate that an OI with the same machine type as the revoked SI is used for re-execution.

Assume that task t_i is allocated to an SI r_l, and that it starts execution at the time point ST_i. When $X_l - ST_i$ is greater than the required execution time w_i/s_l, t_i can normally finish. Otherwise, its execution is interrupted at X_l, and then it has to be restarted on an OI. Therefore, t_i's execution time on a cloud computation resource r_l can be calculated as

$$ET_{i,l} = \begin{cases} w_i/s_l, \\ \quad \text{if } r_l \text{ is OI or } (r_l \text{ is SI and } X_l - ST_i \geqslant w_i/s_l); \\ w_i/s_l + X_l - ST_i, \\ \quad \text{if } r_l \text{ is OI and } X_l - ST_i < w_i/s_l \end{cases} \tag{2}$$

Assume that tasks t_i and t_j are deployed on r_l and r_m, respectively, and that there is a dependency $e_{i,j}$ with a data amount of $d_{i,j}$ between them. The transfer time of $e_{i,j}$ depends on $d_{i,j}$ and the bandwidth b, and it becomes zero when r_m and r_n refer to the same resource, as shown in the following:

$$TT_{i,j} = \begin{cases} d_{i,j}/b, & \text{if } r_l \neq r_m; \\ 0, & \text{if } r_l = r_m \end{cases} \tag{3}$$

Let RT_l be the earliest time at which resource r_l is ready to execute t_i. Task t_i can start execution on r_l when r_l and the data from its parents are ready, and thus the start time and finish time of task t_i on r_l can be calculated as follows:

$$ST_i = \max\left\{RT_l, \max_{t_j \in t_i\text{'s parents}} \{FT_j + TT_{i,j}\}\right\} \tag{4}$$

$$FT_i = ST_i + ET_{i,l} \tag{5}$$

Let S be a schedule solution for a workflow, which specifies how each task in the workflow is allocated to a certain time slice of computation resources for execution. The makespan of a workflow is its overall execution time and based on S, it can be calculated as

$$M(S) = \max_{t_i \in V} \{FT_i\} \tag{6}$$

Let $\gamma(S)$ represent all the used resources in S, and the total required cost can be acquired via

$$C(S) = \sum_{r_l \in \gamma(S)} c_l \tag{7}$$

The makespan $M(S)$ and total cost $C(S)$ are two crucial metrics when users deploy a workflow in the cloud. A general way for users to address the tradeoff between them is to minimize the usage cost under a deadline constraint. Conventional studies, which merely consider stable cloud resources, usually set a hard deadline constraint that a schedule solution has to meet. However, for volatile resources, $M(S)$ and $C(S)$ are random variables, and a hard constraint means that it needs to be met in all cases, including some extreme ones, which is inconsistent with the user's motivation of using SIs for reducing costs. By contrast, with a stochastic setting, it is much more reasonable and practical to set a soft deadline constraint, indicating that the probability of constraint satisfaction should be maximized.

To have an effective evaluation of S, the execution of S is simulated by sampling the random arrival time of interruption and recording the makespan, cost, and whether the user-specified deadline is met each time. Given that the simulation is repeated N times, the average cost $\overline{C}(S)$ and deadline satisfaction ratio $R(S)$ of a solution S are calculated. Then, a utility function is introduced to obtain an overall evaluation of S in terms of the cost and deadline constraint satisfaction, as shown below:

$$U(S) = R(S)^{\mu}/\overline{C}(S) \tag{8}$$

where $\mu \in (0, \infty)$ is the user's preference degree for constraint satisfaction, and a greater value indicates a higher preference. Maximizing the utility function signifies that the deadline constraint is met with a high probability and the expected cost is low. Next, we present a stochastic scheduling method to optimize this utility function.

4 Methodology

Workflow scheduling is a well-known NP-hard problem[46, 47]. This section presents the scheduling method MCLS for deadline-constrained cost optimization in volatile SIs. In the rest of this section, we present an interruption-oblivious stochastic list scheduling heuristic to build a schedule solution. Then, we evaluate a solution with known interruption time points in runtime. We also provide the details of MCLS.

4.1 Stochastic list scheduling heuristic

List scheduling is a widely used heuristic for workflow scheduling when the execution time of tasks on resources is fixed and known a priori[10, 12, 13]. It consists of two phases: construct a task list by assigning task priorities and allocate each task from this list to the resources in sequence. Below, we introduce two methods to generate stochastic task sequences, and then present the detail for resource allocation.

4.1.1 Task list generation

Several task properties have been proposed to rank tasks in a workflow, such as the upward rank and static level[5]. We choose to use the probabilistic upward rank ρ_i from Ref. [12] to order tasks as it is aware that data transmission time may become zero when calculating the transfer time in Eq. (3). Specifically, it is defined as follows:

$$\rho_i = \max_{t_j \in t_i\text{'s children}} \{\rho_j + \eta_j \times d_{i,j}/b\} + w_i/s^* \tag{9}$$

where s^* is the processing capability of the fastest computation resource and η_j is a Boolean variable denoting whether the transmission time to t_j is considered when calculating ρ_i. It is defined as

$$\eta_j = \begin{cases} 0, & \text{if } 1 - \varphi^{-ccr_j} < \text{rand}\,(); \\ 1, & \text{otherwise} \end{cases} \tag{10}$$

where rand( ) is a function returning a random number in $[0, 1)$, φ is a parameter larger than 1, and ccr_j is the computation to the communication ratio of t_j, i.e., $(w_j/s^*)/(d_{i,j}/b)$. Hence, a smaller ccr_j indicates that the probability that η_j returns 0 is greater, and vice versa. Because ρ_i may be different for each invocation, our first method to construct a stochastic task sequence is ordering tasks based on ρ_i.

To enhance the search space and yield a good solution for MCLS, we introduce the second stochastic task sequence generation method. Precisely, it is designed based on Kahn's topological sort algorithm to ensure that the precedency dependencies between tasks are respected, and its detail is shown in Algorithm 1.

Algorithm 1 Topological sort-based list generation

```
Input: workflow graph
Output: task list $L$
1: $L \leftarrow \varnothing, M \leftarrow \varnothing$;
2: Add $t_{\text{entry}}$ into $M$;
3: while $M \neq \varnothing$ do
4:     $t_i \leftarrow$ choose and remove a task from $M$ randomly;
5:     Add $t_i$ to the tail of $L$;
6:     for each edge $e_{i,k}$ from $t_i$ to $t_k$ do
7:         Remove edge $e_{i,k}$ from the graph;
8:         if $t_k$ has no incoming edge then
9:             Add $t_k$ into $M$;
10:        end if
11:    end for
12: end while
13: return $L$
```

In Algorithm 1, M represents the set of ready tasks. A task is ready when it has no precedence dependencies from other tasks (i.e., incoming edges). In the beginning, only the entry task t_{entry} is ready. When M is not empty, a task t_i from M is randomly chosen and then removed from M (Line 4). It is then appended to L, and all its outgoing edges are removed (Lines 5–7). For t_i's child tasks whose incoming edges have all been removed, they also become ready and are added to M (Lines 8–10). By doing this, the precedency dependencies are respected in L.

4.1.2 Resource allocation

In the resource allocation step, each task from the list is sequentially allocated to a resource by comparing all the candidate resources. Although the computation resources offered by the cloud provider are claimed to be infinite, it is unnecessary to try each of them for resource selection as the unused resources of the same type can be regarded as the same. Hence, the resource candidates that need to be considered include all the resources that have been used in the current solution and those that have not been used but can be added at any time (one resource for each type).

For traditional workflow scheduling models aiming to optimize the makespan, the resource allocation criterion is usually set to select a resource from all the resource candidates, which allows the earliest finish time (e.g., HEFT[13] and PEFT[48]). The considered model aims to achieve deadline-constrained cost optimization, and the resource allocation step should be aware of the deadline constraint and cost. To this end, the user-specified deadline D for the workflow is first distributed to each workflow task, and the criterion for selecting a resource for task t_i is updated as follows: meet t_i's sub-deadline and minimize the cost increment of adding t_i. Specifically, the sub-deadline of each task is calculated based on ρ_i via

$$\delta_i = D \times \frac{\rho_{\text{entry}} - \rho_i + w_i/s^*}{\rho_{\text{entry}}} \tag{11}$$

When no candidate resource meets the sub-deadline, the selection criterion turns to minimizing the finish time of the task instead. By doing so, the probability for the whole workflow to meet the overall deadline is increased.

For brevity, the stochastic list scheduling heuristic employing a ρ_i-based task ordering before resource allocation is denoted as SLS-I, and the one employing a topological sort based ordering is denoted as SLS-II. A topological sort based ordering has greater uncertainty than a ρ_i-based one, as the latter's uncertainty only comes from whether to calculate transfer time.

4.2 Solution evaluation with the predicted interruption time

Given the predicted interruption time Ω in runtime for the used SIs in a schedule solution S, the actual makespan $M(S)$ and cost $C(S)$ for the solution can be acquired. The detail is given in Algorithm 2, where the actual start time and finish time of each task are calculated individually based on the task list (Lines 1–6). For each task, the actual start time AST_i is first calculated via Eq. (4) based on the actual finish time of its parent tasks and the actual ready time of the resource r_l where it is allocated (Lines 2&3).

When $AST_i + w_i/s_l$ is less than the predicted interruption time X_l of SI r_l obtained from Ω, r_l can finish executing t_i without interruption. Otherwise, the execution is interrupted, and t_i is moved to an OI for re-execution. If there is an OI that has been launched and is currently free and it is not slower than r_l, t_i is moved to this OI; otherwise, a new OI is launched for re-executing t_i. Then, t_i's execution time and its actual finish time AFT_i are calculated via Eqs. (2) and (5), respectively. After traversing all the tasks in a workflow, its actual makespan and cost are calculated and returned.

4.3 MCLS

The main procedure of MCLS follows the Monte Carlo simulation framework, a popular approach for various problems which involve stochastic factors and are generally infeasible to resolve via deterministic computations. MCLS consists of two main phases, that is, "producing" and "selecting". In the producing phase, an independent sample is randomly taken from the domain space for each random variable X_l, a list scheduling method is leveraged to generate a schedule solution based on random samples, and the above steps are repeated until adequate candidate solutions are generated. Then, in the selecting phase, candidate solutions are evaluated, and the best one is returned. The whole algorithm of MCLS is described in Algorithm 3.

First, an empty solution pool P is created, and SLS-I is performed to obtain a schedule solution S, which is added to P (Lines 1&2). In the producing phase with N_p iterations, a sample of X_l is taken for all used SIs in S based on their distribution functions, which is denoted as Ω, and then S is evaluated based on Ω via Algorithm 2 to compute its makespan and cost values (Lines 3&4).

Next, SLS-II is performed to produce a new solution S'. The makespan and cost of S' are calculated based on Ω. A new solution S' is regarded as better than S if they both meet the deadline constraint and S' incurs less cost or if one of them does not meet the deadline and the makespan of S' is less than S. S' is added into P if it is better than S. Based on the samples Ω, N_d new solutions are built and evaluated, and this procedure is repeated for N_p times.

In the selecting phase, when $|P| > P_{\min}$, N_s samples are taken from their distribution functions to evaluate the candidate solutions in P (Lines 14–17). Then, for each solution, its satisfaction rate and average cost are calculated, and then the utility value is obtained (Line 18). Then, solutions in P whose ranks in terms of utility value are lower than $|P|/2$ are removed. By doing

Algorithm 2 Evaluate a solution

```
Input: solution $S$, predicted interruption time $\Omega$
Output: actual makespan and cost
1: for each task $t_i$ in a task list do
2:     Get the resource $r_l$ in $S$ where $t_i$ is allocated;
3:     Calculate $t_i$'s actual start time $AST_i$ via Eq. (4);
4:     Get the interruption time $X_l$ of $r_l$ from $\Omega$;
5:     Calculate $t_i$'s execution time via Eq. (2), and actual finish
       time $AFT_i$ via Eq. (5);
6: end for
7: Calculate $M(S)$ via Eq. (6) and $C(S)$ via Eq. (7);
8: return $M(S)$ and $C(S)$
```

Algorithm 3 MCLS

```
Input: workflow and distribution function of $X_l$
Output: schedule solution
1: Create an empty solution pool $P$;
2: Perform SLS-I to obtain solution $S$, and add it into $P$;
3: while $N_p \to 0$ do
4:     Take a sample of $X_l$ for used SIs in $S$, denoted as $\Omega$;
5:     Evaluate $S$ based on $\Omega$;
6:     while $N_d \to 0$ do
7:         Perform SLS-II to obtain $S'$;
8:         Evaluate $S'$ based on $\Omega$;
9:         Add $S'$ into $P$ if $S'$ is better than $S$;
10:    end while
11:    Reset $N_d$;
12: end while
13: while $|P| > P_{\min}$ do
14:    while $N_s \to 0$ do
15:        Take a sample of $X_l$ for used SIs, denoted as $\Omega$;
16:        Evaluate each solution $S$ in $P$ based on $\Omega$;
17:    end while
18:    Calculate utility values of solutions in $P$ via Eq. (8);
19:    Sort solutions in $P$ according to utility ascendingly;
20:    Remove solutions from $P$ with a rank $\leqslant |P|/2$;
21:    Reset $N_s$;
22: end while
23: return solution in $P$ with the highest utility value
```

so, half of the solutions in P survive the evaluation in the next round. When $|P| \leqslant P_{\min}$, the solution with the highest average utility value is returned as the final solution. The selection phase guarantees that the performance of the returned solution is robust toward various execution situations in runtime as it is the best one validated by a large number of samplings and evaluations.

MCLS builds $N_p \times N_d$ schedule solutions in total. To build each solution via list scheduling, the computational complexity is $O(n \times (n+t))$, where n is the number of tasks in a workflow and t is the number of resource types. As t is usually much less than n, it can be simplified as $O\left(n^2\right)$. The computational complexity of the producing phase is $O(N_p \times N_d \times n^2)$. The computational complexity of the selecting phase is $O(\log_2|P| \times |P| \times N_s \times n)$ because P can be cut for $\log_2|P|$ times at most, and an $O(n)$ time complexity is required for a solution evaluation. Combining the above analysis, the computational complexity of MCLS can be deduced, i.e., $O\left(N_p \times N_d \times n^2 + \log_2|P| \times |P| \times N_s \times n\right)$.

5 Performance Evaluation

This section reports the experiments conducted to validate the proposed method and discusses the key findings.

5.1 Experimental setup

Experiments were conducted to evaluate the performance of MCLS[49]. Specifically, experiments were conducted on a PC with Intel Core i7 2.9 GHz, 16 GB RAM, Windows 10, and Java 10. We tested the performance of MCLS through extensive evaluations using four realistic scientific workflows, i.e., Montage, Epigenomics, CyberShake, and SIPHT. These workflow applications differ in terms of structure, communication data, and computational characteristics, and they were derived from the Pegasus project, which publishes several realistic scientific workflow applications. Figure 2 illustrates the general structures of these workflows.

Because there is no existing method that completely handles the same problem as this study, we mainly adopted three deadline-constrained cost optimization approaches for comparison, namely, IC-PCP[8], L-ACO[12], and W-LA[10], which are originally designed for OIs, and a robust workflow scheduling approach, i.e., SDLS[44], which schedules precedence constrained stochastic tasks. Precisely, they were adapted to use SIs, and when an SI interruption occurred, an OI was used for resubmission. Moreover, the original L-ACO, which only uses OIs, was also compared (denoted as OD). The parameters of these methods were set as suggested in the literature. The parameters for MCLS were set based

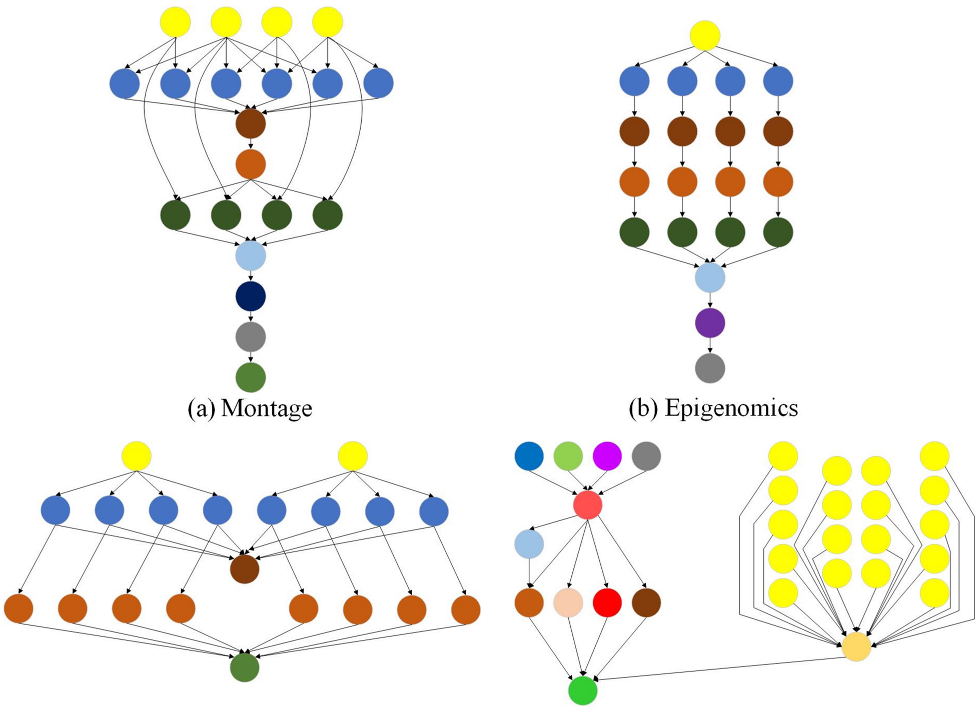

Fig. 2 Structure of four real-world workflow applications.

on parameter tunings: $P_{\min}$ is 4, N_p is 20, N_d is 40, and N_s is 100. Moreover, the Mersenne Twister was adopted to simulate pseudorandom numbers for Monte Carlo simulations. In our experiments, we assumed the use of a data center offering nine types of cloud resources with different processing capacities and costs. Each type of cloud resource was provisioned in the on-demand and spot pricing models. In the current SI market, SI prices are smooth and predictable and do not change in the short term. Hence, we consider a fixed cost for SIs, and the SI price (denoted as $cost_{SI}$) is one-fourth of that of OIs in the same type (denoted as $cost_{OI}$). A long-term price prediction mechanism can be directly incorporated, and the rationale of experiments remains the same. The details of the cloud resources are listed in Table 1, where the speed we used to characterize the resources is represented as a multiple of the speed of the standard resource, because tasks in the above datasets are characterized by runtime on a standard computation resource instead of task workloads. The average bandwidth between computation resources was set to 500 MBps, which is the approximate average bandwidth in Amazon EC2 after it announced to increase the instance network bandwidth. The time interval to charge was set to 1 hour, also the same as that in Amazon EC2.

SIs may be recycled by cloud providers at any time, and we assume that the SI interruption follows an exponential distribution with a rate parameter as it is the maximum entropy probability distribution for a random variable, which is not less than zero and with a fixed expected value, and it is also the most widely used model for describing the inter-arrival time length. Although the rate parameter may change in a long-term trend, it can be regarded as a fixed value in the short term. For the sake of simplicity and focus, we introduce the parameter λ to represent the expected number of interruptions when executing a workflow W on SIs. Let L be the total execution time of all tasks in W with the slowest type of cloud resource. Accordingly, the rate parameter of the exponential distribution is λ/L.

Table 1 Capabilities and costs of available resource types.

Type	Speed	$Cost_{OI}$ (\$)	$Cost_{SI}$ (\$)
Type_1	1.0	0.120	0.030 00
Type_2	1.5	0.195	0.048 75
Type_3	2.0	0.280	0.070 00
Type_4	2.5	0.375	0.093 75
Type_5	3.0	0.480	0.120 00
Type_6	3.5	0.595	0.148 75
Type_7	4.0	0.720	0.180 00
Type_8	4.5	0.855	0.213 75
Type_9	5.0	1.000	0.250 00

To specify a deadline for each workflow, we introduce two benchmarks: (1) cheap schedule, where only one OI of the cheapest type is used, the workflow is scheduled by the HEFT, and the resultant makespan is denoted as M_C; and (2) fast schedule, where only the fastest type of OIs is used, the workflow is also scheduled by the HEFT, and the resultant makespan is denoted as MF. We introduce the deadline factor $\beta(\beta \in [0,1])$ to represent the looseness degree of deadlines, and the deadline of a workflow is determined based on β as shown in the following:

$$D = M_F + (M_C - M_F) \times \beta \tag{12}$$

Due to the randomness of experiments, all the approaches were repeated 100 times in each case, and the average result is reported here.

5.2 Results

To test the performance of MCLS in different settings, we first fixed the parameters λ to 1.0 and β to 0.03, and set μ to 0.5, 1, and 2. The utility, SR, and cost values of each approach for each workflow are listed in Table 2, where the best result of utility in each case is highlighted in bold.

In Table 2, the utility, SR, and cost values vary largely for different workflows. For example, the utility value of each approach is less than 0.1 for Montage but may be larger than 1 for Epigenomics, and the cost is less than 2 for Epigenomics but larger than 10 for Montage and CyberShake. The OD achieves the highest SR among all the peers, but it incurs a very high cost because of the use of OIs. Its utility value is the lowest for all workflows except SIPHT, verifying that it is quite effective to improve the user utility by using OIs and SIs. The SDLS performs similarly to OD: a high SR value and a high cost. The utility values of L-ACO and IC-PCP are higher than those of the SDLS and OD but lower than that of MCLS. MCLS achieves the highest utility value in each case, indicating that it achieves the best tradeoff between the SR and cost. Concretely, MCLS attains the lowest monetary cost and a relatively high success ratio. When $\mu = 2$, it obtains an SR value only 5% lower than that of the OD on average, but the cost is reduced by 74% in return. Moreover, with the increase in μ for MCLS, the SR arises with the expense of the cost value.

Then, we fixed the parameters μ to 1 and β to 0.005, and varied λ from 0.2 to 3.0 at a step of 0.2. The utility

Table 2 Utility, SR, and cost of each approach with respect to μ.

Application	Method	$\mu = 0.5$			$\mu = 1.0$			$\mu = 2.0$		
		Utility	SR	Cost ($)	Utility	SR	Cost ($)	Utility	SR	Cost ($)
Montage	MCLS	**0.054**	0.95	18.075	**0.053**	0.97	18.054	**0.051**	0.98	18.114
	L-ACO	0.051	0.95	19.254	0.052	0.98	19.974	0.048	0.96	19.084
	IC-PCP	0.044	1.00	22.963	0.044	1.00	22.773	0.044	1.00	22.912
	W-LA	0.042	0.99	23.418	0.044	1.00	22.960	0.043	1.00	23.012
	SDLS	0.017	1.00	59.146	0.017	1.00	59.417	0.017	1.00	59.500
	OD	0.014	1.00	71.973	0.014	1.00	73.155	0.014	1.00	72.095
Epigenomics	MCLS	**2.465**	0.91	0.387	**2.239**	0.92	0.441	**2.191**	0.94	0.403
	L-ACO	2.217	0.91	0.430	2.049	0.92	0.449	2.017	0.94	0.438
	IC-PCP	2.029	0.92	0.473	1.909	0.93	0.487	1.964	0.97	0.479
	W-LA	1.794	0.92	0.535	1.642	0.91	0.554	1.641	0.94	0.539
	SDLS	1.048	0.97	0.940	1.015	0.97	0.956	0.993	0.97	0.948
	OD	0.658	1.00	1.520	0.646	1.00	1.548	0.650	1.00	1.548
CyberShake	MCLS	**0.078**	0.93	12.44	**0.079**	0.94	11.894	**0.082**	0.96	11.208
	L-ACO	0.067	0.87	13.842	0.068	0.90	13.232	0.068	0.94	13.019
	IC-PCP	0.067	0.96	14.574	0.064	0.90	13.965	0.068	0.97	13.799
	W-LA	0.063	0.91	15.246	0.062	0.89	14.291	0.061	0.93	14.115
	SDLS	0.022	0.98	45.917	0.021	0.95	45.896	0.220	0.99	45.208
	OD	0.021	1.00	47.992	0.021	1.00	48.512	0.021	1.00	48.403
SIPHT	MCLS	**0.507**	0.78	1.741	**0.557**	0.84	1.509	**0.481**	0.85	1.502
	L-ACO	0.463	0.72	1.831	0.498	0.79	1.586	0.401	0.80	1.594
	IC-PCP	0.344	0.79	2.586	0.335	0.84	2.505	0.283	0.84	2.491
	W-LA	0.441	0.79	2.017	0.476	0.84	1.764	0.410	0.84	1.720
	SDLS	0.050	0.88	18.700	0.048	0.89	18.508	0.041	0.87	18.585
	OD	0.210	0.99	4.740	0.208	0.98	4.710	0.214	1.00	4.679

value of each method for different workflows is depicted in Fig. 3. OD is not included here as it is only used for OIs. The first impression we can derive from Fig. 3 is that with the increase of λ, the utility value in each case generally falls down with a little fluctuation. This is because when there are more interruptions during the execution of a workflow, the required makespan and cost become larger, and the SR value becomes lower. For each workflow and each case of λ, MCLS achieves the highest utility value among all the methods, indicating that it performs the best in terms of the required cost and success rate.

Afterward, we varied β from 0.01 to 0.05 with a step of 0.005, and the experimental result is depicted in Fig. 4. The SDLS is not included here as it is oblivious to β. In Fig. 4, the utility value rises up with the increase in β in most cases. That is because when the deadline becomes looser, the required cost declines, and the success rate grows. In general, MCLS outperforms L-ACO, and the latter outperforms IC-PCP in terms of utility value. For example, when β is 0.02, the utility value of MCLS is close to 2.0 for Epigenomics, whereas those of L-ACO and IC-PCP are lower than 1.8. The results verify that MCLS can find more competitive solutions with a high success rate and low cost in volatile cloud resources.

6 Conclusion

In this paper, we present a robust workflow scheduling method called MCLS to minimize the execution cost under a user-specified soft deadline for SIs in cloud environments. Based on a Monte Carlo simulation framework, the MCLS employs a sampled task execution time to produce candidate solutions based on deadline distribution and list scheduling. It returns the most robust one as the output from the solution pool with a specific evaluation mechanism and selection criteria. Experimental results show that the MCLS achieves higher utility values than some traditional algorithms. The results also verify that it is very promising to combine SIs with OIs for executing workflows, as the monetary cost is largely reduced at the expense of a slight decrease in the success ratio. In the future, we plan to design scheduling techniques for running more diverse applications in SIs, e.g., bag-of-tasks. We are also interested in designing long-term scheduling strategies for SIs where the long-term price change of SIs should be considered.

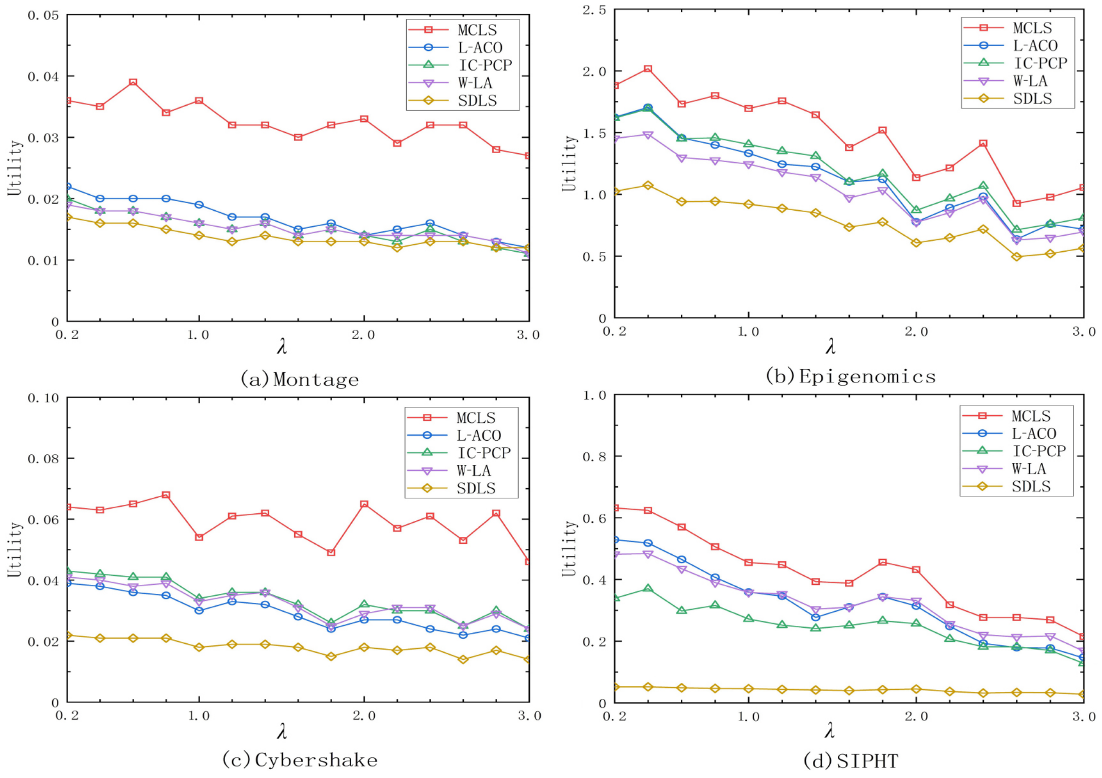

Fig. 3 Utility value of each approach with respect to λ.

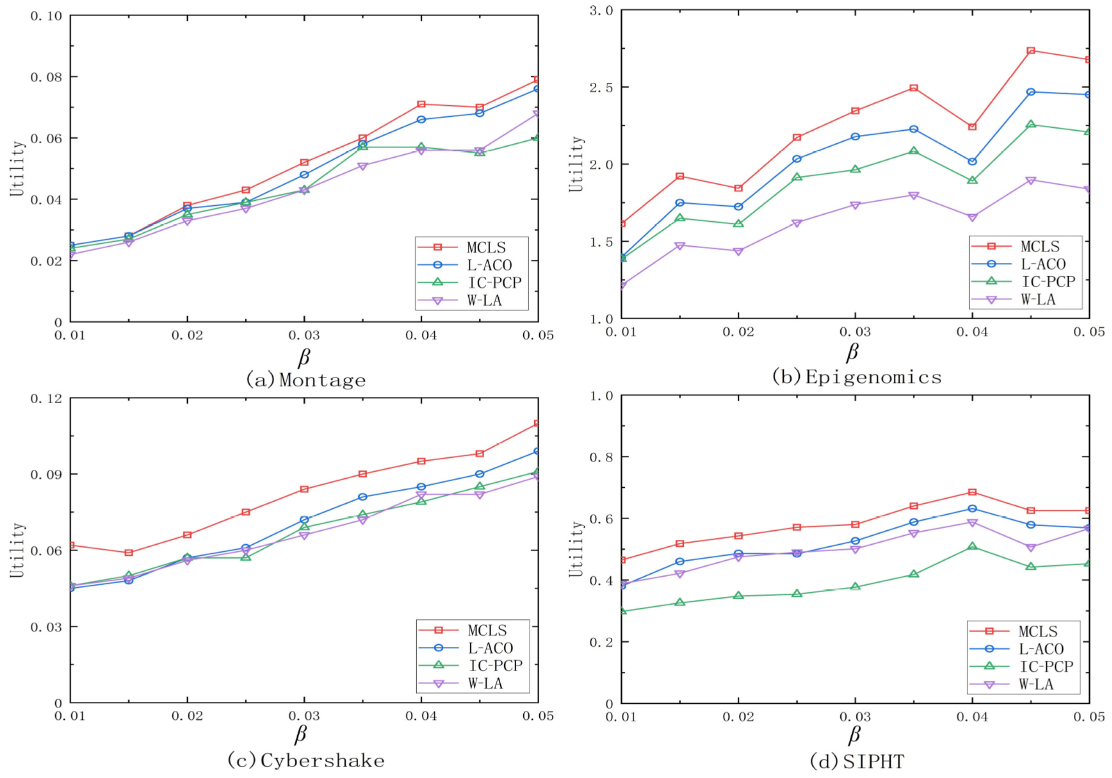

Fig. 4 Utility value of each approach with respect to β.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (Nos. 62172065 and 62072060) and the Natural Science Foundation of Chongqing (No. cstc2020jcyj-msxmX0137).

References

[1] J. Sahni and D. P. Vidyarthi, A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment, *IEEE Trans. Cloud Comput.*, vol. 6, no. 1, pp. 2–18, 2018.

[2] L. Zheng, C. Joe-Wong, C. W. Tan, M. Chiang, and X. Y. Wang, How to bid the cloud, *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 71–84, 2015.

[3] B. Javadi, R. K. Thulasiram, and R. Buyya, Characterizing spot price dynamics in public cloud environments, *Future Gener. Comput. Syst.*, vol. 29, no. 4, pp. 988–999, 2013.

[4] S. Mandal, G. Maji, S. Khatua, and R. K. Das, Cost minimizing reservation and scheduling algorithms for public clouds, *IEEE Trans. Cloud Comput.*, doi: 10.1109/TCC.2021.3133464.

[5] M. Adhikari, T. Amgoth, and S. N. Srirama, A survey on scheduling strategies for workflows in cloud environment and emerging trends, *ACM Comput. Surv.*, vol. 52, no. 4, p. 68, 2019.

[6] D. Poola, K. Ramamohanarao, and R. Buyya, Enhancing reliability of workflow execution using task replication and spot instances, *ACM Trans. Auton. Adapt. Syst.*, vol. 10, no. 4, p. 30, 2016.

[7] D. W. Wei, H. S. Ning, F. F. Shi, Y. L. Wan, J. B. Xu, S. K. Yang, and L. Zhu, Dataflow management in the internet of things: Sensing, control, and security, *Tsinghua Science and Technology*, vol. 26, no. 6, pp. 918–930, 2021.

[8] S. Abrishami, M. Naghibzadeh, and D. H. J. Epema, Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds, *Future Gener. Comput. Syst.*, vol. 29, no. 1, pp. 158–169, 2013.

[9] V. Arabnejad, K. Bubendorfer, and B. Ng, Budget and deadline aware e-science workflow scheduling in clouds, *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 1, pp. 29–44, 2019.

[10] L. W. Yang, L. J. Ye, Y. Q. Xia, and Y. F. Zhan, Look-ahead workflow scheduling with width changing trend in clouds, *Future Gener. Comput. Syst.*, vol. 139, pp. 139–150, 2023.

[11] M. A. Rodriguez and R. Buyya, Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds, *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 222–235, 2014.

[12] Q. W. Wu, F. Ishikawa, Q. S. Zhu, Y. N. Xia, and J. H. Wen, Deadline-constrained cost optimization approaches for workflow scheduling in clouds, *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 12, pp. 3401–3412, 2017.

[13] H. R. Faragardi, M. R. S. Sedghpour, S. Fazliahmadi, T. Fahringer, and N. Rasouli, GRP-HEFT: A budget-constrained resource provisioning scheme for workflow scheduling in IaaS clouds, *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1239–1254, 2020.

[14] H. Topcuoglu, S. Hariri, and M. Y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, 2002.

[15] R. Ghafouri, A. Movaghar, and M. Mohsenzadeh, A budget constrained scheduling algorithm for executing workflow application in infrastructure as a service clouds, *Peer-to-Peer Netw. Appl.*, vol. 12, no. 1, pp. 241–268, 2019.

[16] N. Rizvi and D. Ramesh, Fair budget constrained workflow scheduling approach for heterogeneous clouds, *Cluster Comput.*, vol. 23, no. 4, pp. 3185–3201, 2020.

[17] J. J. Durillo and R. Prodan, Multi-objective workflow scheduling in Amazon EC2, *Cluster Comput.*, vol. 17, no. 2, pp. 169–189, 2014.

[18] Q. W. Wu, M. C. Zhou, Q. S. Zhu, Y. N. Xia, and J. H. Wen, MOELS: Multiobjective evolutionary list scheduling for cloud workflows, *IEEE Trans. Autom. Sci. Eng.*, vol. 17, no. 1, pp. 166–176, 2020.

[19] X. M. Zhou, G. X. Zhang, J. Sun, J. L. Zhou, T. Q. Wei, and S. Y. Hu, Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based HEFT, *Future Gener. Comput. Syst.*, vol. 93, pp. 278–289, 2019.

[20] L. Dierks and S. Seuken, Cloud pricing: The spot market strikes back, *Manage. Sci.*, vol. 68, no. 1, pp. 105–122, 2022.

[21] O. A. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir, Deconstructing Amazon EC2 spot instance pricing, *ACM Trans. Econ. Comput.*, vol. 1, no. 3, p. 16, 2013.

[22] G. J. Portella, G. N. Rodrigues, E. Y. Nakano, A. Boukerche, and A. C. M. Melo, A novel statistical and neural network combined approach for the cloud spot market, *IEEE Trans. Cloud Comput.*, doi: 10.1109/TCC.2021.3091936.

[23] J. Li, Y. M. Zhu, J. D. Yu, C. N. Long, G. T. Xue, and S. Y. Qian, Online auction for IaaS clouds: Towards elastic user demands and weighted heterogeneous VMs, *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 9, pp. 2075–2089, 2018.

[24] W. Voorsluys and R. Buyya, Reliable provisioning of spot instances for compute-intensive applications, in *Proc. 2012 IEEE 26th Int. Conf. Advanced Information Networking and Applications*, Fukuoka, Japan, 2012, pp. 542–549.

[25] X. He, P. Shenoy, R. Sitaraman, and D. Irwin, Cutting the cost of hosting online services using cloud spot markets, in *Proc. 24th Int. Symp. High-Performance Parallel and Distributed Computing*, Portland, OR, USA, 2015, pp. 207–218.

[26] S. Yang, S. Khuller, S. Choudhary, S. Mitra, and K. Mahadik, Scheduling ML training on unreliable spot instances, in *Proc. 14th IEEE/ACM Int. Conf. Utility and Cloud Computing Companion*, Leicester, UK, 2021, p. 29.

[27] L. Teylo, A. L. Nunes, A. C. M. A. Melo, C. Boeres, L. M. de A. Drummond, and N. F. Martins, Comparing SARS-CoV-2 sequences using a commercial cloud with a spot instance based dynamic scheduler, in *Proc. 2021 IEEE/ACM 21st Int. Symp. Cluster, Cloud and Internet Computing (CCGrid)*, Melbourne, Australia, 2021, pp. 247–256.

[28] F. Xu, H. Y. Zheng, H. Jiang, W. J. Shao, H. K. Liu, and Z. Zhou, Cost-effective cloud server provisioning for predictable performance of big data analytics, *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 5, pp. 1036–1051, 2019.

[29] S. J. Cao, K. F. Deng, K. J. Ren, X. Y. Li, T. F. Nie, and J. Q. Song, An optimizing algorithm for deadline constrained scheduling of scientific workflows in IaaS clouds using spot instances, in *Proc. 2019 IEEE Int. Conf. Parallel and Distributed Processing with Applications, Big Data and Cloud Computing, Sustainable Computing and Communications, Social Computing and Networking* (*ISPA/BDCloud/SocialCom/SustainCom*), Xiamen, China, 2019, pp. 1421–1428.

[30] R. G. Martinez, A. Lopes, and L. Rodrigues, Planning workflow executions when using spot instances in the cloud, in *Proc. 34th ACM/SIGAPP Symp. Applied Computing*, Limassol, Cyprus, 2019, pp. 310–317.

[31] H. Ghavamipoor, S. A. K. Mousavi, H. R. Faragardi, and N. Rasouli, A reliability aware algorithm for workflow scheduling on cloud spot instances using artificial neural network, in *Proc. 2020 10th Int. Symp. Telecommunications* (*IST*), Tehran, Iran, 2020, pp. 67–71.

[32] G. George, R. Wolski, C. Krintz, and J. Brevik, Analyzing AWS spot instance pricing, in *Proc. 2019 IEEE Int. Conf. Cloud Engineering* (*IC2E*), Prague, Czech Republic, 2019, pp. 222–228.

[33] A. C. Zhou, J. M. Lao, Z. B. Ke, Y. Wang, and R. Mao, FarSpot: Optimizing monetary cost for HPC applications in the cloud spot market, *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 11, pp. 2955–2967, 2022.

[34] L. Teylo, L. Arantes, P. Sens, and L. M. A. Drummond, A dynamic task scheduler tolerant to multiple hibernations in cloud environments, *Cluster Comput.*, vol. 24, no. 2, pp. 1051–1073, 2021.

[35] T. P. Pham and T. Fahringer, Evolutionary multi-objective workflow scheduling for volatile resources in the cloud, *IEEE Trans. Cloud Comput.*, vol. 10, no. 3, pp. 1780–1791, 2022.

[36] F. Cao and M. X. Zhu, A fault-tolerant workflow mapping algorithm under end-to-end delay constraint, in *2011 IEEE Int. Conf. High Performance Computing and Communications*, Banff, Canada, 2011, pp. 575–580.

[37] H. Youness, A. Omar, and M. Moness, An optimized weighted average makespan in fault-tolerant heterogeneous MPSoCs, *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 8, pp. 1933–1946, 2021.

[38] G. S. Yao, Y. S. Ding, and K. R. Hao, Using imbalance characteristic for fault-tolerant workflow scheduling in cloud systems, *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 12, pp. 3671–3683, 2017.

[39] D. Poola, S. K. Garg, R. Buyya, Y. Yang, and K. Ramamohanarao, Robust scheduling of scientific workflows with deadline and budget constraints in clouds, in *Proc. 2014 IEEE 28th Int. Conf. Advanced Information Networking and Applications*, Victoria, Canada, 2014, pp. 858–865.

[40] J. Zhou, Y. Zhang, and W. F. Wong, Fault tolerant stencil computation on cloud-based GPU spot instances, *IEEE Trans. Cloud Comput.*, vol. 7, no. 4, pp. 1013–1024, 2019.

[41] L. C. Canon and E. Jeannot, Evaluation and optimization of the robustness of DAG schedules in heterogeneous environments, *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 4, pp. 532–546, 2010.

[42] W. Zheng and R. Sakellariou, Stochastic DAG scheduling using a Monte Carlo approach, *J. Parallel Distrib. Comput.*, vol. 73, no. 12, pp. 1673–1689, 2013.

[43] X. Y. Tang, K. L. Li, G. P. Liao, K. Fang, and F. Wu, A stochastic scheduling algorithm for precedence constrained tasks on grid, *Future Gener. Comput. Syst.*, vol. 27, no. 8, pp. 1083–1091, 2011.

[44] K. L. Li, Li, X. Y. Tang, B. Veeravalli, and K. Q. Li, Scheduling precedence constrained stochastic tasks on heterogeneous cluster systems, *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 191–204, 2015.

[45] T. P. Pham, S. Ristov, and T. Fahringer, Performance and behavior characterization of amazon EC2 spot instances, in *Proc. 2018 IEEE 11th Int. Conf. Cloud Computing* (*CLOUD*), San Francisco, CA, USA, 2018, pp. 73–81.

[46] H. Wang, L. Cai, X. Hao, J. Ren, and Y. H. Ma, ETS-TEE: An energy-efficient task scheduling strategy in a mobile trusted computing environment, *Tsinghua Science and Technology*, vol. 28, no. 1, pp. 105–116, 2023.

[47] Z. Y. Hu and D. S. Li, Improved heuristic job scheduling method to enhance throughput for big data analytics, *Tsinghua Science and Technology*, vol. 27, no. 2, pp. 344–357, 2022.

[48] H. Arabnejad and J. G. Barbosa, List scheduling algorithm for heterogeneous systems by an optimistic cost table, *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 3, pp. 682–694, 2014.

[49] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, *Softw.: Pract. Exper.*, vol. 41, no. 1, pp. 23–50, 2011.

Quanwang Wu received the BEng, MEng, and PhD degrees in computer science from Chongqing University, China in 2007, 2010, and 2013, respectively. He was a special researcher at the Digital Content and Media Sciences Research Division of the National Institute of Informatics (NII) in Tokyo, Japan from 2014 to 2015. He is currently an associate professor at Chongqing University, Chongqing, China. His main research interests include service-oriented computing, cloud computing, and data mining.

Jianzhao Fang received the BEng degree from Harbin University of Science and Technology, China in 2017. He is currently a master student in computer science and engineering at Chongqing University. His main research interests include cloud computing, computational intelligence, and data mining.

Jie Zeng received the MEng degree from Chongqing University, China in 2016. She is currently a researcher at the National Experimental Teaching Demonstration Center, Chongqing University, China. Her main research interests include environmental data analysis and intelligent scheduling.

Junhao Wen received the PhD degree from Chongqing University, China in 2008, where he is a professor at the College of Big Data and Software Engineering. His research interests include service computing, cloud computing, and software dependable engineering. He has published over 80 refereed journal and conference papers in the above areas. He has over 30 research and industrial grants, and developed many commercial systems and software tools.

Fengji Luo received the BEng and MEng degrees in software engineering from Chongqing University, China in 2006 and 2009, respectively, and the PhD degree in electrical engineering from The University of Newcastle, Australia in 2014. Currently, he is a lecturer and academic fellow at the School of Civil Engineering, The University of Sydney, Australia. His research interests include smart grid, energy informatics, and computational intelligence.