A delay-based dynamic scheduling algorithm for bag-of-task workflows with stochastic task execution times in clouds

Zhicheng Cai[a,d,e,*], Xiaoping Li[b], Rubén Ruiz[c], Qianmu Li[a]

[a]*School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China*
[b]*School of Computer Science and Engineering, Southeast University, Nanjing, China*
[c]*Instituto Tecnológico de Informática, Acc. B. Universitat Politècnica de València, València, Spain*
[d]*Key Laboratory of Image and Video Understanding for Social Safety, Nanjing, China*
[e]*Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education*

Abstract

Bag-of-Tasks (BoT) workflows are widespread in many big data analysis fields. However, there are very few cloud resource provisioning and scheduling algorithms tailored for BoT workflows. Furthermore, existing algorithms fail to consider the stochastic task execution times of BoT workflows which leads to deadline violations and increased resource renting costs. In this paper, we propose a dynamic cloud resource provisioning and scheduling algorithm which aims to fulfill the workflow deadline by using the sum of task execution time expectation and standard deviation to estimate real task execution times. A bag-based delay scheduling strategy and a single-type based virtual machine interval renting method are presented to decrease the resource renting cost. The proposed algorithm is evaluated using a cloud simulator ElasticSim which is extended from CloudSim. The results show that the dynamic algorithm decreases the resource renting cost when guaranteeing the workflow deadline compared to the existing algorithm.

Keywords: cloud computing, scheduling, workflow, bag of tasks, stochastic

1. Introduction

A Bag-of-Tasks (BoT) consists of many independent tasks which can be processed in parallel [1]. BoTs are widely spread in many fields such as image processing, parameter sweeping and data mining. Applications of these fields

*Corresponding author.
Email addresses: `caizhicheng@njust.edu.cn` (Zhicheng Cai), `xpli@seu.edu.cn` (Xiaoping Li), `rruiz@eio.upv.es` (Rubén Ruiz), `qianmu@njust.edu.cn` (Qianmu Li)

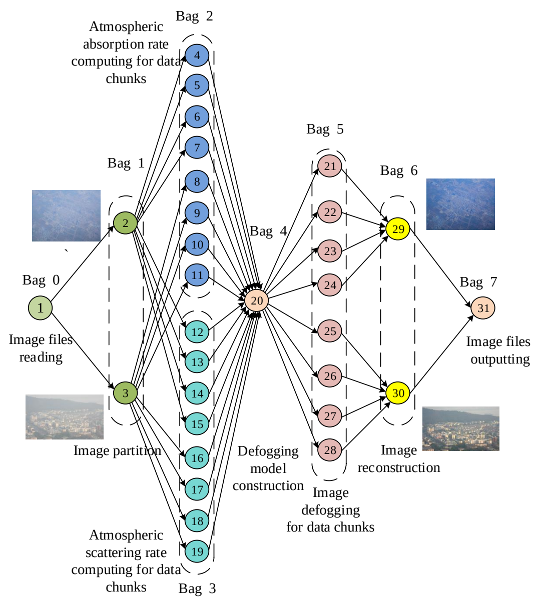

Bag 0
Bag 1
Bag 2
Bag 3
Bag 4
Bag 5
Bag 6
Bag 7
Atmospheric absorption rate computing for data chunks
Image files reading
Image partition
Atmospheric scattering rate computing for data chunks
Defogging model construction
Image defogging for data chunks
Image reconstruction
Image files outputting

Figure 1: An example of Bag-of-Task based workflow applications for image defogging

usually consist of several (sequential, parallel or hybrid of sequential and parallel) steps rather than a single step and each step processes a BoT [2]. Data files are transferred among these BoTs which forms a BoT workflow that implements complex business logic [3]. Figure 1 shows an example of BoT workflows for an image defogging application for two image files [4]. The application consists of eight steps: image files reading, image partition of each image file, atmospheric absorption rate computing for data trunks, atmospheric scattering rate computing for data chunks, defogging model construction, image defogging for data chunks, image reconstruction and image files outputting. Each step usually processes many independent data chunks which form a BoT and the workflow is composed of eight BoTs. When the number of processed image files becomes larger, the number of tasks at each BoT becomes larger too.

Running BoT workflows requires a large number of computing resources, such as grid clusters, supercomputers and private clusters. Nowadays cloud computing has become an available platform for many applications which provides resources on demand to improve application performance and to reduce the costs by renting only the needed resources. When BoT workflow applications are migrated to public clouds, resource provisioning and scheduling algorithms are needed. They decide the time, the type of resources and the number of resources to rent in order to reduce as much as possible the renting cost. In this paper, we aim to minimize the cloud resource renting cost of a BoT workflow with deadline constraints, in which task execution times are stochastic. The problem of optimal scheduling tasks with precedence relationships, in general cases, is

NP-hard [5]. For example, the problem to minimize the cost of workflows under deadline constraints considered by Möhring et al. [6], and Demeulemeester et al. [7] can be modeled as the Discrete Time-Cost Tradeoff Problem (DTCTP), which has been proved to be NP-hard [8]. In this paper, the problem to minimize the cost of BoT workflows under deadline constraints with stochastic task execution times and interval-based cloud resource pricing models is much more complex than DTCTP. However, most of existing elasticity mechanisms provided by commercial public clouds are designed for web-server based applications and few of them focus on computation-intensive workflow applications [9].

Different from a single BoT or multiple independent BoTs, there are complex dependencies among different BoTs of the same BoT workflow. In the academic work, most existing scheduling algorithms of BoT applications are designed for a single BoT or multiple independent BoTs [10, 11, 12]. These methods are not suitable for BoT workflows with many connected and constrained BoTs, for example the workflow shown in Figure 1.

Most existing workflow scheduling algorithms are designed according to deterministic task execution times by assuming that task execution times can be predicted accurately [3, 13, 14, 15, 16, 17, 18]. However, practical task execution times are stochastic and have different probability distributions because of Virtual Machine (VM) performance uncertainty and complicated task properties [19]. When algorithms based on deterministic task execution times are put into practice, algorithm performances degenerate because of task execution time uncertainty [20], e.g., deadlines are violated or additional resource rental costs are incurred. Two strategies are usually adopted to deal with task execution time uncertainty [21]: (i) Static algorithms tolerate to some extent task execution time uncertainty [22, 23, 24] and (ii) Dynamic algorithms which schedule tasks according to real time execution states as much as possible [20]. In these methods, task execution times with different probability distributions are usually fixed to known values. The maximum task execution times are usually adopted, which however overestimates the practical task execution times and leads to resource over provisioning and additional costs. The sum of task execution time expected value and standard deviation is used by Tang et al. [24] for task scheduling with stochastic task execution times on private clusters, obtaining a good performance.

Most existing static workflow scheduling methods considering task execution time uncertainty are designed to minimize workflow makespans on clusters with fixed capacities, which are not suitable for the considered cloud resource rental cost minimization problem. In this paper, cloud resources are rented and released dynamically by intervals during the workflow runtime (called resource runtime auto-scaling). Scheduled tasks are more likely to exceed the planned resource interval horizons, which incurs an additional resource renting cost. In other words, resource runtime auto-scaling makes the workflow scheduling with task execution time uncertainty more complex. Therefore, developing static algorithms which are robust for task execution time uncertainty for BoT workflows is much more complicated.

Dynamic algorithms are widely used to handle task execution time uncertainty. However, the performance of dynamic algorithms is usually worse than that of static algorithms because of myopic optimization and a lack of complete information [20]. Therefore, it is crucial to determine when to make resource renting decisions for a dynamic algorithm. For the dynamic algorithms developed by Malawskiet et al. [20], VM renting decisions are made whenever the utilization is above or below given thresholds and it has been proved that making dynamic algorithms aware of workflow structures is beneficial for improving performance. However, the algorithms proposed by Malawskiet et al. [20] are tailored for maximizing the number of completed workflows rather than minimizing the rental cost of a single BoT workflow.

In this paper, a delay-based dynamic (online) algorithm is proposed to deal with stochastic task execution times. The main contributions are the following:

(1) A bag-based delay triggering strategy for VM renting process is proposed to fully use the bag structure to improve the performance of the dynamic algorithm. Tasks are delayed and not scheduled until tasks of the whole BoT are ready, i.e., virtual machine renting decisions are only made when tasks of a whole BoT are ready in order to decrease the resource rental cost of the whole BoT.

(2) An expectation-and-variance based VM selection method is proposed to handle the task execution time uncertainty. The sum of task execution time expected value and standard deviation is adopted to estimate the practical task execution times on VMs properly, which is benefical for improving the utilization of rented VM intervals.

(3) A single-type based greedy method for VM renting of each ready BoT is developed to improve the effectiveness and efficiency simultaneously. Resource provisioning for each BoT under deadline constraints at each step of the dynamic algorithm is NP-hard. Since tasks of the same BoT have the same function, they have the same virtual machine performance requirements, the type and the amount of required VMs are determined based on the assumption that only a single type of VMs are provisioned.

The rest of this paper is organized as follows. Section 2 gives an overview of the related work. Section 3 presents the descriptions of the cloud environment and BoT workflow applications. Section 4 describes the details of our proposed dynamic algorithm for dealing with stochastic task execution times. Performance evaluation is discussed in Section 5. Finally, conclusions and future work are outlined in Section 6.

2. Related work

Resource scheduling is one of the most important issues when applications are migrated to clouds [9, 25]. Many factors (such as horizontal and vertical scalability of capacity, various resource pricing models, uncertainty of task execution times and resource performances) make the cloud resource provisioning and scheduling complex. There are many scheduling algorithms for cloud computing with different objectives, such as cloud resource renting cost [14, 16, 18, 26], task finish time [15, 27], energy consumption [28, 29], the number of finished workflow instances [30]. Different algorithms are designed for different types of applications which consist of different types of tasks: such as web requests, bag of tasks, Map-Reduce tasks, workflows and BoT workflows. Workflow applications are widespread in many fields and attract much more attention. Therefore, they are considered in this paper.

There are some scheduling methods which assume that the rented resources are kept unchanged during the whole workflow runtime. Although provisioning with unchanged resources decreases complexity, renting resource intervals dynamically during the workflow runtime is beneficial for decreasing resource rental cost [31]. Therefore, workflow scheduling methods are classified into two types according to the resource renting flexibility: (i) One is to rent appropriate capacity of resources which are kept unchanged during the workflow runtime. For example, Byun et al. [26] proposed the Balanced Time Scheduling (BTS) algorithm to determine the rented type and number of virtual machines during the whole workflow runtime. Based on fixed capacity of resources, Chen et al. [15] proposed a clustering method to minimize the workflow execution time and Verma et al. [32] developed a heuristic to minimize the execution time and cost simultaneously. (ii) The other one is to rent resources by intervals dynamically during the workflow runtime. For example, algorithms which minimize the resource rental cost by dynamically renting resource intervals were developed by Byun et al. [14], Abrishami et al. [16], Durillo et al. [17], Li et al. [13] and Su et al. [33], etc.

Most existing workflow scheduling algorithms only give a static resource renting and task assignment plan based on deterministic task execution times in advance. Estimating task execution times is the basis of workflow scheduling and there are some task execution time prediction methods [34, 35, 36, 37]. However, deviations between predicted task execution times and actual task execution times are unavoidable [19]. When these static algorithms are put into practice, the practical execution states are different from the original schedule because of task execution time uncertainty and task failures, among other issues [38]. As introduced by Malawski et al. [20], the stochastic task execution times have a great impact on the workflow resource provisioning when static algorithms are put into practice.

Many algorithms have considered stochastic task execution times for workflow scheduling, but they are for workflows with fixed capacities of resources (such as private data centers and data centers composed of fixed number of virtual machines rented from public clouds). A method which guarantees the mathematical expectation of the workflow makespan was proposed by Skutella et al. [22]. Kamthe et al. [23] developed a stochastic scheduling method to minimize the workflow makespan expectation. Zheng et al. [21] developed a Monte Carlo method to select schedules that tolerate stochastic task execution times to minimize the workflow makespan. The mathematical expectation and standard deviation of task execution times was adopted by Tang et al. [24] to minimize the workflow makespan. These algorithms minimize the workflow makespan on data centers with fixed capacities, which are not suitable for cost minimization. In this paper, we furthermore consider dynamically rented resource intervals.

Few of the workflow scheduling methods have considered the resource runtime auto-scaling and task execution time uncertainty simultaneously. Rodriguez et al. [39] proposed a Particle Swarm Optimization (PSO) method to rent resource intervals dynamically for workflows with stochastic task execution times, in which the maximum task execution time was adopted to deal with the stochastic task execution time. The PSO was not tailored for wotkflows with BoT structures. Malawski et al. [20] aimed to maximize the number of finished workflows considering both resource runtime auto-scaling and stochastic task execution times which is different from the rental cost minimization of a single BoT workflow considered in this paper.

Most existing scheduling algorithms are designed for independent BoTs or single-task based workflows and few of them consider BoT workflows. For example, cloud resources are dynamically rented for independent BoTs to minimize resource rental cost or makespan [10, 11, 12]. A MapReduce-workflow scheduling algorithm was developed by Tang et al. [40] which was designed for applications on heterogeneous computing platforms with fixed capacities rather than workflows on cloud computing with runtime auto-scaling resources. Wang et al. [3] and Cai et al. [18] proposed several algorithms for deadline or budget constrained BoT workflow scheduling problems. However, Wang et al. [3] considered a special type of BoT workflows, which are composed of sequential BoTs. The algorithms proposed by Cai et al. [18] are job (BoT)-level based, which assume that tasks of the same job (BoT) must be processed at the same time. In this paper, tasks of the same job (BoT) are processed asynchronously to improve the utilization of rented resource intervals although the job (BoT)-based structure information is still used to optimize the problem.

In a word, most existing workflow scheduling algorithms are either designed for independent BoTs, are based on deterministic task execution times or are tailored for resources with fixed capacities. In this paper, we propose a method that surmounts all these weakness and considers rich BoT structures, stochastic task execution times and dynamic rented resources.

3. Problem description

In the following sections, we describe the considered problem in detail.

3.1. Cloud resources

Different types of virtual machines are provided by public clouds. Interval-based pricing models are offered and the whole interval (month, hour or minute) is paid even if only a part of the interval is used. Let P_m be the price of VM m (per interval unit) and $\mathbf{L}_m$ be the length of the pricing interval of VM m. The VM setup time can not be ignorable since a VM is not immediately available after the request is sent. T^l_m represents the VM setup time (loading) of VM m.

3.2. Bag-of-Task based workflow applications

A BoT workflow can be represented by a Directed Acyclic Graph (DAG) with bag structures $G = \{\mathcal{B}, V, E\}$. $\mathcal{B} = \{B_0, B_1, \ldots, B_Q\}$ is the set of bags of tasks, in which B_i is the i^{th} bag of tasks. $V = \{v_1, \ldots, v_N\}$ is the set of tasks of all BoTs and $\mathcal{B}$ is a partition of V. $E = \{(i,j)|i < j\}$ is the precedence constraints of tasks where (i,j) indicates that v_j cannot start until v_i completes. $\mathcal{P}_i$ represents the immediate predecessor set of v_i. For the example in Figure 1, $V = \{v_1, \ldots, v_{23}\}$, $\mathcal{B} = \{B_0, B_1, \ldots, B_6\}$, $B_0 = \{v_1\}$, $B_1 = \{v_2, v_3\}$, $B_2 = \{v_4, v_5, \ldots, v_{11}\}$, $B_3 = \{v_{12}, v_{13}, \ldots, v_{19}\}$, $B_4 = \{v_{20}\}$, $B_5 = \{v_{21}, v_{22}, \ldots, v_{28}\}$, $B_6 = \{v_{29}, v_{30}\}$ and $B_7 = \{v_{31}\}$.

As mentioned above, task execution times are stochastic because of VM performance uncertainty and task properties complexity [19, 24]. The probability distribution types of task execution times might be uniform or normal, among others. Probability distribution of task execution times can be estimated by several types of methods [34, 35], which is outside the scope of this paper. Besides the execution time of each task, data transfer times among tasks are dependent on the volume of data and the system network bandwidth W. We assume that a hybrid storage architecture is applied which consists of a global and a local storage system. The global storage system is a distributed file system on dedicated infrastructures, e.g., Amazon S3. The local storage system is the file system of each rented VM, which can only be accessed by the VM itself. Produced data files are written into both of the global storage system and the local file system simultaneously. When a VM is released, files stored in its local file system are deleted. If a task is assigned to a VM which contains some required data files in the local storage system, data transfer times of these files are zero. Otherwise, the required data files are needed to be loaded from the global storage system. Professional softwares is required to be installed before tasks can be processed, which consume significant software setup times.

3.3. Resource provisioning and workflow scheduling

As shown in Figure 2, a workflow management system usually consists of: (1) A Workflow Engine which is in charge of monitoring the real time states and events of workflow execution. It is invoked whenever a task is finished, a rented VM is available or a rented interval is finished. (2) A Workflow Scheduler which decides when, which type of and how many VMs should be rented according to the applied scheduling algorithm. It also decides when to release VM instances and how to assign tasks to VMs. (3) A Virtual Data Center which is composed of different types of VMs rented dynamically by intervals from public clouds. Assumptions for the considered cloud workflow system are as follows:

(1) There is no limitation on the number of rented VM instances since the number of required VMs for each workflow instance is usually smaller than the limit numbers of commercial public clouds.

(2) A VM can only process one task at a time while other assigned tasks are blocked and waiting in a queue. The reason is that applications can be coded by taking advantage of multi-threading, i.e., the VM processor cores are

assumed to be fully used at the same time. Processing multiple tasks simultaneously requires additional context switching which might not accelerate the processing speed.

(3) A task cannot be preempted after it is assigned to a VM for processing.

There are deadlines for many BoT workflow applications. The objective of this paper is to minimize the resource rental cost while fulfilling the deadline constraint D. Let $f_{i,m}$ be the execution time probability distribution of task v_i on VM m. $I_{i,k}$ and $\mathcal{I}_i$ represent the k-th input data file and the set of input files of task v_i respectively. $Z_{i,k}$ is the size of file $I_{i,k}$. If $I_{i,k}$ is at the local file system of VM m at time t, $x_{i,k,m,t} = 1$. Otherwise, $x_{i,k,m,t} = 0$. ϖ_i and $T^s_{\varpi_i}$ represent the needed professional software type and its setup time for task v_i. If ϖ_i has been already installed on VM m and it is present at time t, $y_{i,m,t} = 1$. Otherwise, $y_{i,m,t} = 0$. T^b_m and T^f_m represent the start renting time (available time at which the created VM is just returned from the provider) and release time of VM m respectively. ζ is the sorted set of all rented VMs which are sorted in the non decreasing order of start renting times. m_i denotes the VM of ζ to which v_i is assigned. T^b_{i,m_i}, T^e_{i,m_i} and T^f_{i,m_i} are the start processing time, execution time and finish time of v_i on m_i respectively. $\gamma_{(m,a)}$ denotes the index of the a-*th* assigned task on VM m. The mathematical model is as follows:

$$\min \sum_{m\in\zeta} \frac{T^f_m - T^b_m}{\mathbf{L}_m} \times P_m \tag{1}$$

st.

$$T^f_{i,m_i} = T^b_{i,m_i} + T^e_{i,m_i} + \frac{\sum_{I_{i,k}\in\mathcal{I}_i}(1 - x_{i,k,m_i,T^b_{i,m_i}})Z_{i,k}}{W} + (1 - y_{i,m_i,T^b_{i,m_i}})T^s_{\varpi_i}, v_i \in V \tag{2}$$

$$T^e_{i,m_i} = \mathit{Sampling}(f_{i,m_i}), v_i \in V \tag{3}$$

$$T^b_{i,m_i} \geq \max_{v_j\in\mathcal{P}_i}\{T^f_{j,m_j}\}, v_i \in V \tag{4}$$

$$T^b_{\gamma_{(m,a+1)},m} \geq T^f_{\gamma_{(m,a)},m}, m \in \zeta, a \in \{1, \ldots, A_m - 1\} \tag{5}$$

$$T^f_{\gamma_{(m,A_m)},m} \geq T^f_m, m \in \zeta \tag{6}$$

$$T^b_{\gamma_{(m,1)},m} \geq T^b_m, m \in \zeta \tag{7}$$

$$T^f_m = T^b_m + n \times \mathbf{L}_m, n \in N^+, m \in \zeta \tag{8}$$

$$T^f_{N,m_N} \leq D \tag{9}$$

The objective function (1) minimizes the resource rental cost of all VMs. The processing time of a task consists of a task execution time, data transfer times and a software setup time according to Constraint (2). Constraint (3) illustrates that practical task execution times are samples of task execution time probability distributions and Constraint (4) guarantees that tasks start

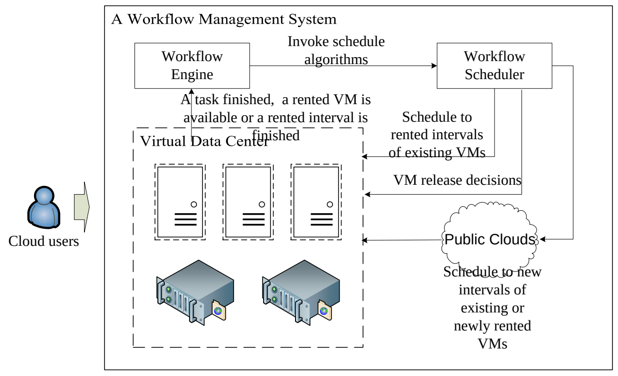

A Workflow Management System
Workflow Engine
Invoke schedule algorithms
Workflow Scheduler
A task finished, a rented VM is available or a rented interval is finished
Schedule to rented intervals of existing VMs
Virtual Data Center
VM release decisions
Cloud users
Public Clouds
Schedule to new intervals of existing or newly rented VMs

Figure 2: Architecture of a workflow management system

after their predecessor tasks are finished. According to Constraints (5), (6) and (7), tasks on each VM are processed one by one and task execution times are within the horizon of rented intervals. Constraint (8) illustrates that VMs are rented dynamically by intervals. The workflow deadline D is fulfilled by Constraints (9).

4. Proposed delay-based dynamic scheduling algorithm

Developing dynamic algorithms is an effective method to deal with task execution time uncertainty. The reason is that dynamic algorithms can make decisions according to the real time conditions. However, dynamic algorithms can not get good performance because of myopic optimization and a lack of complete information. It has been shown that the performance of dynamic algorithms can be improved by using global information [20]. In this paper, a bag-based deadline division and a bag-based delay scheduling methods are proposed to use global information as much as possible in order to improve the effectiveness while maintaining the efficiency. The proposed Delay-based Dynamic Scheduling (DDS) algorithm is composed of three main steps: (1) Bag-based deadline division, which divides the workflow deadline into BoT deadlines. (2) Bag-based delay scheduling, which decides which tasks should be scheduled immediately and which tasks should be delayed and (3) A single-type based resource renting, which rents appropriate type and number of VMs considering the whole BoT.

4.1. Bag-based deadline division

Task deadlines determine the degree of task execution parallelism which has a great impact on the utilization of rented VM intervals. Dividing workflow deadlines into task deadlines before workflow execution considering global workflow information is beneficial to improve algorithm performance. Many existing deadline division methods do not consider the bag structure of BoT-workflows

[13, 16] which is however helpful to divide the workflow deadline properly. In Cai et al. [41], a Bag-based Deadline Division method (BDD) is proposed which considers the number of tasks and execution times of bags. BDD defines the Estimated Wasted Cost (EWC) of each bag to be the wasted cost of rented intervals on a give execution duration. Larger EWC means higher possibility of additional cost. For each bag, giving longer processing durations usually leads to smaller EWC. There are different functions of the EWC to the processing duration for distinct bags. The problem of Distributing the workflow deadline to competitive bags to minimize the total EWC can be modeled as a Discrete Time-Cost Tradeoff Problem (DTCTP) which is NP-hard [41]. The return-rate of each bag is defined as the ratio of decreased EWC to the consumed duration. A Largest Return-Rate First heuristic method (LRRF) is used in the BDD method to distribute the workflow deadline to bags. Details of the BDD are provided in Cai et al. [41] which are omitted here. Let D_{v_i} be the deadline of task v_i. For a BoT workflow, the BDD is called only one time at the beginning of the proposed dynamic algorithm. Let M be the number of VM types provided by cloud providers and N be the number of workflow tasks. The complexity of BDD is $O(MN^3)$ [41].

4.2. Bag-based delay scheduling

The proposed dynamic algorithm is invoked to schedule new tasks whenever a scheduled task has been finished. Ready tasks are updated by adding tasks whose predecessors have finished. If tasks are scheduled as soon as ready, the result is a myopic resource renting plan. Such approach is overly greedy. However, if more global information is used, fewer VMs and more appropriate VM types will be used A BoT workflow consists of multiple BoTs and tasks of the same BoT have the same function and VM performance requirements. Renting VMs for a large number of tasks of the same bag together has a possibility to decrease the total resource rental cost. Figures 3 and 4 show an example of how the resource renting cost can be decreased by considering the BoT as a whole. $v_1, v_2, \ldots, v_6$ belong to the same BoT and become ready at three different times (two for each release time). $m4.xlarge$ and $c4.2xlarge$ are two types of VMs with the same pricing interval length of 60 minutes. The prices per interval of $m4.xlarge$ and $c4.2xlarge$ are 0.239 \$ and 0.419 \$ respectively. In Figure 3, resource intervals are rented at each release time, which greedily minimizes the added cost considering only the current ready tasks. For the six tasks, three intervals are rented in the end and the total cost is 0.717 \$. In Figure 4, tasks are put into a wait queue at the first two release times. Resource renting plans are made when all tasks of the BoT are ready at the third release time. Finally, only one interval is needed and the cost is 0.419 \$ which is much cheaper than 0.717 \$ However, waiting more tasks to finish and make VM plan according to a batch of tasks (delay scheduling) will delay task executions. When the task deadline is very tight, delay scheduling will lead to short task execution times which limits the scheduler to choose fast and expensive VM types. Delay scheduling will, on the contrary, increase VM rental cost. Therefore, delaying tasks to take advan-

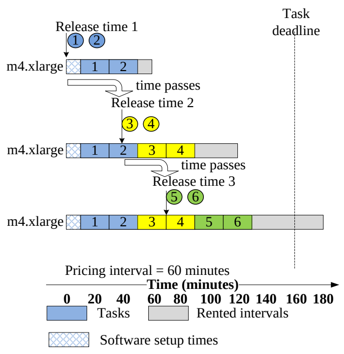


Figure 3: An example of resource renting strategies which make renting plans whenever there are ready tasks

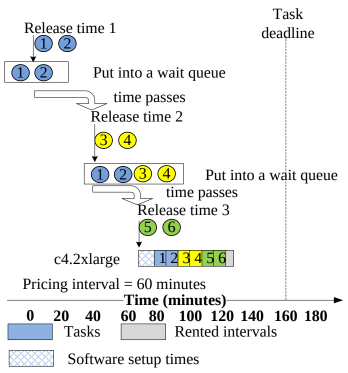


Figure 4: An example of renting strategies which make renting plans considering the BoT as a whole

tage of global information and left task execution times should be balanced.

In this paper, a bag-based delay trigger strategy for the VM renting process is proposed. Workflow scheduling with the bag-based delay trigger strategy is referred to as bag-based delay scheduling. Because there are idle times on previous rented intervals, ready tasks are scheduled so long as there are enough idle times on rented intervals first. When tasks are scheduled to idle times of rented intervals, it is crucial to decide which VM to select. The estimation of task execution times is the basis for VM selection. The expected value and standard deviation of task execution times have been evaluated and a good performance was observed in many stochastic scheduling problems [24, 42]. Therefore, the sum of task execution time expectation and standard deviation is used to estimate actual task execution times in this paper. Then, left ready tasks will be checked whether they should be delayed or scheduled to new VM intervals. Under this delay trigger strategy, a task v_i will be delayed (not scheduled), if the ratio *rate* of ready tasks to the total number of tasks of bag $B_k(v_i \in B_k)$ is smaller than a delay threshold α. Otherwise, a resource renting plan is made based on a batch of tasks of the bag together. A larger α means that the scheduler will wait more tasks to be ready and make resource renting plan based on a larger number of tasks. When $rate = 1$, a task can be scheduled only when all tasks of its bag are ready.

Algorithm 1: Bag-based Delay Scheduling (BDS)

```
Input: ready task set $\Gamma$, delay threshold $\alpha$, created VM set $\zeta^c$, set $\zeta^r$ of VMs being
       created
begin
    Initialize $\Gamma^d \leftarrow \varnothing$, $\Gamma^s \leftarrow \varnothing$;
    foreach $v_i \in \Gamma$ do
        $m^s \leftarrow null$, $F^e \leftarrow +\infty$;
        foreach $m^a \in \zeta^c$ do
            Calculate $C_{i,m^a}$ and $F_{i,m^a}$ according to Equations (12) and (13) ;
            if $C_{i,m^a} = 0$ and $F_{i,m^a} < F^e$ then
                $m^s \leftarrow m^a$, $F^e \leftarrow F_{i,m^a}$ ;
        foreach $m^c \in \zeta^r$ do
            Calculate $C_{i,m^c}$ and $F_{i,c}$ according to Equations (12) and (13);
            if $C_{i,m^c} = 0$ and $F_{i,m^c} < F^e$ then
                $m^s \leftarrow m^c$, $F^e \leftarrow F_{i,m^c}$ ;
        if $m^s \neq null$ then
            Send task $v_i$ to $m^s$ for processing;
            $\Gamma \leftarrow \Gamma - \{v_i\}$;
            $T^a_{m^s} \leftarrow T^a_{m^s} + Q_{i,m^s_i}$;
        else
            $N_r \leftarrow$Calculate the number of ready tasks of bag $B_k(v_i \in B_k)$;
            $rate \leftarrow (N_r/|B_k|) \times 100$;
            $N_l \leftarrow$Calculate the number of unready tasks of bag $B_k(v_i \in B_k)$;
            if $rate > \alpha | N_l = 0$ then
                $\Gamma^s \leftarrow \Gamma^s \cup \{v_i\}$;
            else
                $\Gamma^d \leftarrow \Gamma^d \cup \{v_i\}$;
    return $\Gamma^d$ and $\Gamma^s$;
```

Details of the bag-based delay scheduling method are described in Algorithm 1. ζ^c and ζ^r represent the set of created VMs (which are available at present) and VMs being created (of which renting requests have been sent to VM providers but that are not still available at the present) respectively. At first, ready tasks (whose predecessors have been finished) are sorted by estimated shortest task execution times in a descending order. Then, tasks are tested one by one to idle times of rented intervals on VMs in ζ^c and ζ^r. For each VM m in ζ^c and ζ^r, T^b_m and T^a_m represent the start renting time and total execution time of assigned tasks. The estimated start time of the current task v_i on m is

$$S_{i,m} = T^b_m + T^a_m \tag{10}$$

The estimated total processing time of v_i on m is

$$\begin{aligned} Q_{i,m} = \quad & E[p_{i,m}] + \sqrt{V(f_{i,m})} + (1 - y_{i,m,S_{i,m}})T^s_{\varpi_i} \\ & + (\textstyle\sum_{I_{i,k} \in \mathcal{I}_i} (1 - x_{i,k,m,S_{i,m}})Z_{i,k})/W \end{aligned} \tag{11}$$

in which $E[f_{i,m}]$ and $V(f_{i,m})$ are the expected value and variance of $f_{i,m}$, respectively. The estimated finish time of v_i on m is calculated by

$$F_{i,m} = S_{i,m} + Q_{i,m} \tag{12}$$

The increased resource renting cost of assigning v_i on m is calculated by

$$C_{i,m} = \Big(\Big\lceil \frac{(T^a_m + Q_{i,m})}{\mathbf{L}_m} \Big\rceil - \Big\lceil \frac{T^a_m}{\mathbf{L}_m} \Big\rceil\Big) P_m \tag{13}$$

If there are VMs with $C_{i,m} = 0$, the VM m^s with the earliest estimated finish time is chosen. Then, v_i is scheduled to m^s directly and the total execution time of assigned tasks on m^s is updated by $T^a_{m^s} \leftarrow T^a_{m^s} + Q_{i,m}$. Otherwise, it is checked that whether v_i should be delayed. N_r is the number of ready tasks of bag B_k and N_l is the number of unready tasks of bag $B_k(v_i \in B_k)$. The ready rate is calculated by $rate \leftarrow N_r/|B_k|$. If $rate > \alpha$ or $N_l = 0$, v_i is added to the set Γ^s which consists of tasks ready to be scheduled to newly rented intervals. Otherwise, v_i is added to the task set Γ^d which is composed of delayed tasks.

In BDS, the time complexity of sorting tasks is $O(N^2)$. There are N iterations at step 4 at most since the number of tasks in the Γ is less than N. Each task of Γ is tested on rented VMs of which the number is N at most (the VM number is not larger than the task number). Therefore, the time complexity of BDS is $O(N^2)$.

4.3. A single-type based resource interval renting method

In BDS of the dynamic algorithm, some tasks are scheduled to idle times of rented intervals and some tasks are delayed. In this stage, tasks of set Γ^s are scheduled to new intervals of existing and (or) new VMs. Tasks can be scheduled to elastic number of VM instances for parallel, sequence or hybrid of parallel-and-sequence execution. In other words, tasks of the same bag are considered together to make VM renting plan but can have different start times. Γ^s usually consists of tasks belong to different bags. Bags are scheduled one by one and the bag with larger total execution time is scheduled earlier. New VM interval renting decisions (The number of parallel VMs, the type of VMs and the number of rented intervals on each VM instance) will be made based on a batch of tasks of the same bag together. Optimal scheduling of tasks to elastic numbers of resources to minimize the cost is still NP-hard [10]. Let N_{B_k} be the number of ready tasks of bag B_k and M be the number of VM types provided by the public cloud. There are $(M + 1)^{N_{B_k}}$ number of VM renting alternatives.

A single-VM-type based resource renting strategy is proposed which assumes that only a single type of VMs can be rented because tasks of the same bag have the same function and the same VM performance requirements. Therefore, only solutions with the same VM type are evaluated, which decreases the search space greatly (there are $M \times N_{B_k}$ VM renting possibilities at most for the single type strategy). The resource renting and scheduling plan of each VM type is obtained by a list based task scheduling method which further decreases the number of searched solutions to M. In the list based scheduling method, tasks are scheduled one by one and each task is scheduled to an existing or a new VM with the cheapest increased cost while fulfilling the task deadline. If there are no VMs fulfilling the task deadline, the VM with the earliest estimated finish time is chosen.

Details of the Single-type Interval Renting method (SIR) are shown in Algorithm 2. Let T_c be the current system time. T^t_m is the total execution time of temporarily assigned tasks on each VM m when evaluating each VM type and is initialized to zero. ζ^p_δ is the set of planned VMs which have been planned to be rented for VM type δ, but the renting requests have not been sent yet. For a given VM type $\delta \in \Omega$ (Ω is the set of VM types), tasks are scheduled to an existing or to a new VM one by one. For a task v_i, the estimated finish time and the increased cost on each VM m of ζ^c_δ, ζ^r_δ and ζ^p_δ are calculated by

$$F_{i,m} = T^b_m + T^{a'}_m + Q_{i,m} \tag{14}$$

and

$$C_{i,m} = (\left\lceil \frac{(T^{a'}_m + Q_{i,m})}{\mathbf{L}_m} \right\rceil - \left\lceil \frac{T^{a'}_m}{\mathbf{L}_m} \right\rceil) \times P_m \tag{15}$$

in which $T^{a'}_m = T^a_m + T^t_m$. ζ^c_δ and ζ^r_δ contain VMs of ζ^c and ζ^r with type δ respectively. The increased cost and the estimated finish time on a new VM m' are also calculated with $T^b_{m'} = T_c$ and $T^a_{m'} = 0$. If there are VMs fulfilling the task deadline, the VM with the cheapest increased cost is chosen. When there are multiple VMs with the same cheapest increased cost, the VM with the earliest start renting time is chosen. This strategy is designed to minimize the number of rented VMs. If there is no VM fulfilling the task deadline, the VM with the earliest finish time (labeled by F_e) is chosen. When there are multiple VMs with the same earliest finish time, the VM with the cheapest increased cost (labeled by C') is selected over the others. If the chosen VM m^s_i is the new VM m', m' is added to ζ^p_δ. The assignment of v_i to m^s_i is recorded in the schedule S_δ. $T^t_{m^s_i}$ is then updated by $T^t_{m^s_i} \leftarrow T^t_{m^s_i} + Q_{i,m^s_i}$. After all tasks are scheduled, the total increased cost $C_{B_k,\delta}$ and the maximum deadline violation $V_{B_k,\delta}$ of VM type δ is calculated by Equation (16) and (17), respectively.

$$C_{B_k,\delta} = \sum_{v_i \in B_k} C_{i,m^s_i} \tag{16}$$

$$V_{B_k,\delta} = \max_{v_i \in B_k} \{F_{i,m^s_i} - D_{v_i}\} \tag{17}$$

The total increased cost and the maximum deadline violation on the next VM type are calculated again as mentioned above. After B_k is checked on all VM types, the VM type δ' with the smallest maximum deadline violation (labeled

Algorithm 2: Single-type Interval Renting (SIR)

```
Input: current time $T_c$, task set $\Gamma^s$, created VM set $\zeta^c$, set $\zeta^r$ of VMs being created
begin
    foreach $B_k \in \Gamma^s$ do
        $\delta^{'} \leftarrow null$, $\zeta^p_{\delta^{'}} \leftarrow \varnothing$, $S_{\delta^{'}} \leftarrow null$, $V^s \leftarrow +\infty$, $C^s \leftarrow +\infty$;
        foreach $m \in \zeta^r$ do
            if $T^b_m < T_c$ then
                $T^b_m \leftarrow T_c$;
        foreach $\delta \in \Omega$ do
            foreach $m \in \zeta^c_\delta \cup \zeta^r_\delta$ do
                $T^t_m \leftarrow 0$;
            $\zeta^p_\delta \leftarrow \emptyset$;
            foreach $v_i \in B_k$ do
                foreach $m \in \zeta^c_\delta \cup \zeta^r_\delta \cup \zeta^p_\delta$ do
                    Calculate $F_{i,m}$ and $C_{i,m}$ according to Equations (14) and (15);
                Add a new VM $m^{'}$, calculate $F_{i,m^{'}}$ and $C_{i,m^{'}}$ according to
                Equations (14) and (15) ( $T^b_{m^{'}} \leftarrow T_c + T^l_m$, $T^a_{m^{'}} \leftarrow 0$ );
                $C^{'} \leftarrow +\infty$, $m^s_i \leftarrow null$;
                foreach $m \in \zeta^c_\delta \cup \zeta^r_\delta$ do
                    if $F_{i,m} \leq D_{v_i}$ and $C_{i,m} < C^{'}$ then
                        $m^s_i \leftarrow m$, $C^{'} \leftarrow C_{i,m}$;
                foreach $m \in \zeta^p_\delta$ do
                    if $F_{i,m} \leq D_{v_i}$ and $C_{i,m} < C^{'}$ then
                        $m^s_i \leftarrow m$, $C^{'} \leftarrow C_{i,m}$;
                if $F_{i,m^{'}} \leq D_{v_i}$ and $C_{i,m^{'}} < C^{'}$ then
                    $m^s_i \leftarrow m^{'}$, $C^{'} \leftarrow C_{i,m^{'}}$;
                if $m^s_i = null$ then
                    $F_e \leftarrow +\infty$, $C^{'} \leftarrow +\infty$;
                    foreach $m \in \zeta^c_\delta \cup \zeta^r_\delta \cup \zeta^p_\delta \cup \{m^{'}\}$ do
                        if $F_{i,m} < F_e$ or ($F_{i,m} = F_e$ and $C_{i,m} < C^{'}$) then
                            $m^s_i \leftarrow m$, $C^{'} \leftarrow C_{i,m}$, $F_e \leftarrow F_{i,m}$;
                if $m^s_i = m^{'}$ then
                    $\zeta^p_\delta \leftarrow \zeta^p_\delta \cup \{m^{'}\}$;
                Record the assignment of $v_i$ to $m^s_i$ in $S_\delta$, $T^t_{m^s_i} \leftarrow T^t_{m^s_i} + Q_{i,m^s_i}$;
            Calculate $C_{B_k,\delta}$ and $V_{B_k,\delta}$ according to Equations (16) and (17);
            if $V_{B_k,\delta} < V^s$ or ($V_{B_k,\delta} = V^s$ and $C_{B_k,\delta} < C^s$) then
                $\delta^{'} \leftarrow \delta$, $\zeta^p_{\delta^{'}} \leftarrow \zeta^p_\delta$, $S_{\delta^{'}} \leftarrow S_\delta$, $V^s \leftarrow V_{B_k,\delta}$, $C^s \leftarrow C_{B_k,\delta}$;
        Rent new VMs and schedule tasks of $B_k$ according to $\zeta^p_{\delta^{'}}$ and $S_{\delta^{'}}$;
        Update $T^b_m \leftarrow T_c$ for each $m$ of $\zeta^p_{\delta^{'}}$, $\zeta^r \leftarrow \zeta^r \cup \zeta^p_{\delta^{'}}$;
        Update $T^a_m$ for all VM of $\zeta^c$ and $\zeta^r$ according to $S_{\delta^{'}}$;
    return
```

by V^s) is chosen. When there are multiple VM types with the same maximum deadline violation, the VM type with the smallest total increased cost (labeled by C^s) is chosen. New VMs are rented according to planned VM set $\zeta^p_{\delta'}$. For each $m \in \zeta^p_{\delta'}$, $T^b_m \leftarrow T_c$. ζ^r is updated by $\zeta^r \leftarrow \zeta^r \cup \zeta^p_{\delta'}$. Tasks of B_k are scheduled according to $S_{\delta'}$. Tasks assigned to created VMs are sent to the VMs directly for processing. There is a processing list on each VM which consists of tasks to be processed. Tasks assigned to VMs that are being creating can not be sent to them immediately. They are stored in a waiting list of each VM and will be sent to the VMs after these VMs are available from the cloud providers. T^a_m of each VM is updated by adding the sum of execution time expectation and standard deviation of all assigned tasks. The remaining BoTs of Γ^s are scheduled at the same way one by one.

Two other auxiliary functions are needed by the proposed Delay-based Dynamic scheduling algorithm (DDS): OnVMavailable and OnIntervalFinish. When a VM m being created is returned from the VM provider and becomes available, OnVMavailable updates $T^b_m \leftarrow T_c$, removes m from ζ^r and adds m to ζ^c. Tasks in the waiting list of m are sent to m for processing. OnIntervalFinish is in charge of deciding when and which intervals to be released. When a rented interval is ready to be finished, the VM is released if there is no task on the VM. Otherwise, the next interval of the VM is rented continuously.

For SIR, the number of bags at step 2 is N at most, the number of VM types at step 4 is M and the number of tasks of each bag is N at most. For each task, it is tested on existing VMs (N at most). Therefore, the time complexity of SIR is $O(MN^3)$.

4.4. Description of proposed Delay-based Dynamic Scheduling (DDS)

The proposed Delay-based Dynamic Scheduling method (DDS) is shown in Algorithm 3. Whenever a task is finished and returned from the assigned VM, the proposed DDS is called by the Workflow Scheduler. DDS updates the ready task set at first. Then, DDS guarantees that the workflow deadline has been divided. If task deadlines have not been generated, BDD is called. Next, BDS is called to schedule tasks to available times of rented intervals immediately, to decide which tasks should be scheduled to new intervals and which tasks should be delayed. Finally, SIR is called to rent new intervals considering the BoT structure. Based on the time complexity analysis of each component, the time complexity of DDS is $O(MN^3)$.

5. Performance Evaluation

DDS is evaluated on a simulator which we call ElasticSim, which in turn is extended from the popular CloudSim [43] by adding support for resource runtime auto-scaling and stochastic task execution time modeling. The introduction and source code of the ElasticSim is presented at the website [44].

Algorithm 3: Delay-based Dynamic Scheduling

```
Input: current time $T_c$, created VM set $\zeta^c$, set $\zeta^r$ of VMs being created
1 begin
2     Update ready task set $\Gamma^s$;
3     if task deadlines have not been generated then
4         Call BDD() to divide the workflow deadline;
5     $(\Gamma^d, \Gamma^s) \leftarrow$ Call BDS($\Gamma^s$, $\zeta^c$, $\zeta^r$);
6     Call SIR($T_c$,$\Gamma^s$,$\zeta^c$, $\zeta^r$);
7     return
```

5.1. Workflow instances and VM types

Workflow instances produced by the Workflow Generator [45] of Bharathi et al. [46] are adopted which include Montage, CyberShake, Epigenomics (Genomic), LIGO and Sipht workflow instances. The number of tasks of each workflow instance belongs to the set {100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 2000, 3000, 4000, 5000, 6000}. For each size of different workflow types, 20 instances are generated. Therefore, there are $5 \times 15 \times 20 = 1500$ workflow instances. Each workflow instance is saved in an XML file, which provides network structures, task names and execution times. As shown in Table 1, the ElasticSim models some types of Amazon EC2 VMs.

In practice, task execution time probability distributions on different types of VMs can be estimated by several methods [34, 35], which are out of the scope of this paper. In the experiments, task execution time distributions on VMs are generated as follows: Because each VM may consists of multiple identical parallel processors, it is assumed that applications are coded taking advantage of multiple threads and task can be divided into equal number of partitions for parallel execution. For each task v_i, the expectation of the number of instructions is defined as the product of its execution time in the xml file and 20000, labeled by $N^i_{v_i}$. Let s^{cpu}_m be the CPU speed of the processor of VM m and n_m be the number of processors of VM m. The expected value of the task execution time is $E[f_{i,m}] = N^i_{v_i}/(s^{cpu}_m \times n_m)$ by considering the parallel processors as related identical parallel machines [47, 48]. For the Amazon EC2 VMs, only the total CPU speed of all processors in Million Instructions Per second (MIPS) are shown in Table 1. Then, we adopted $E[f_{i,m}]$ as the expected value of the task execution time probability distribution $f_{i,m}$. For a given the distribution type ϑ and a given θ^{max}, the parameters and density function of the distribution $f_{i,m}$ can be determined based on the expected value $E[f_{i,m}]$. In the simulation, the standard deviation of the task execution time is calculated based on $f_{i,m}$ and the practical task execution time of v_i on m is generated from $f_{i,m}$ randomly.

5.2. Compared algorithms

Although there are many workflow scheduling algorithms, only a few scheduling algorithms designed for workflows with dynamically rented VM intervals and stochastic task execution times [39]. However, these stochastic methods were not tailored for BoT structures or considered different objectives. In other words,

Table 1: Configurations and Prices (per hour) for Amazon EC2 VMs

VM Type	**Configuration (Memory Size, CPU Speed)**	**Price Per Interval**
Normal types		
m4.L	8000 MB MEM, 3250 MIPS	\$0.12
m4.xL	16000 MB MEM, 6500 MIPS	\$0.239
m4.2xL	32000 MB MEM, 13000 MIPS	\$0.479
High-CPU types		
c4.xL	10000 MB MEM, 10000 MIPS	\$0.299
c4.2xL	15000 MB MEM, 15500 MIPS	\$0.419
c4.2xLe	15000 MB MEM, 22500 MIPS	\$0.489
c4.4xL	30000 MB MEM, 32000 MIPS	\$0.838

there is no scheduling algorithm for the considered BoT workflow with dynamically rented VM intervals and stochastic task execution times. Therefore, we can only compare the proposals with existing algorithms based on deterministic execution times for general or BOT workflows.

Durillo et. al [17] proposed an effective cloud aware Multi-Objective Heterogeneous Earliest Finish Time (MOHEFT) algorithm as an extension to the well-known Heterogeneous Earliest Finish Time (HEFT) algorithm [49] for general workflows. Different from HEFT, the cloud aware MOHEFT (called MOHEFT in brief) rents VM intervals dynamically during the workflow execution horizon. Because MOHEFT schedules tasks based on deterministic execution times, the practical performance when it is adopted to the ElasticSim (for workflows with stochastic task execution times) is different from the static performance. The performance of the proposed DDS heuristics are compared with the practical performance of MOHEFT on the ElasticSim first. The Unit-aware Rules based Heuristic (URH) [41] extended from the Multiple Rules based Heuristic (MRH) [13] is one of the few BoT workflow scheduling algorithms. The URH gets the best performance for BoT workflow scheduling with deterministic task execution times [41]. Therefore, the proposals have been compared with URH too.

When the MOHEFT and URH is put into practice with stochastic task execution times, the practical execution result is different from the static schedule. Therefore, the MOHEFT and URH can not be used in the ElasticSim directly which supports stochastic task execution time modeling and it has been modified as follows: The assignment of tasks to VMs and the sequence of tasks on each VM of static schedule are kept unchanged. However, the start times of tasks can not be directly used because of stochastic task execution times.

DDS uses the sum of task execution time expectation and standard deviation to estimate the stochastic task execution time. The maximum value of a task execution time can also be used. The DDS based on maximum task execution times is called DDS_MAX. α has a great impact on the performance of DDS. In the experiments, α takes values from $\{0, 5, 30, 100\}$, which leads to DDSRR0, DDSRR5, DDSRR30 and DDSRR100 respectively. α of DDS_MAX equals 100.

5.3. Experimental setting

Several factors have great influence on the practical performances of BoT workflow scheduling algorithms, such as the length of cloud resource pricing interval, the type of distribution for the stochastic task execution time, the maximum task execution time prediction deviation and the deadline tightness, among other potential factors. Details of these factors are as follows:

(1) L is the length of Cloud resource pricing interval which has been tested with the values $\{60, 300, 900, 1500, 2100, 2700, 3300, 3600\}$ seconds.

(2) ϑ is the task execution time probability distribution type, which is considered as uniform or normal.

(3) θ^{max} is the maximum percentage of deviation between the predicted task execution time and the practical task execution time tested at values $\{0, 10, 20, 30, 40, 50\}$ (%).

(4) λ is the deadline factor which has been tested with values $\{1.5, 3, 6, 12, 24\}$. The deadline is λ times the shortest workflow makespan D^s, which is obtained by assuming that all tasks are processed on the fastest VMs with maximum degree of parallelism and VMs are ready whenever they are required.

VM setup times are set to be 50 seconds which is approximate to the practical setup times of many popular commercial providers.

We design two groups of experiments. In the first one, DDS algorithms are compared with URH, in which different tightness of deadlines are tested by setting different values to λ. Because MOHEFT is a multi-objective algorithm, DDS and URH algorithms can not be compared with MOHEFT directly. Therefore, in the second group, makespans of MOHEFT solutions are set as the deadlines of URH and DDS algorithm for fairly comparison.

5.4. Comparison with URH

For a workflow instance, URH first gets a stactic schedule ($C^{urh,s}$,$f_{urh,s}$) under a given deadline d. When the URH is put into practice (simulated on ElasticSim), the practical performance is ($C^{urh,p}$,$f_{urh,p}$). For the given deadline d, DDS gets a schedule with performance (C^{dds},f_{dds}). The Percentage of Increased resource rental Cost (PIC) of DDS is defined as $PIC_{dds} = (C^{dds} - C^{urh,s}) \times 100\%/C^{urh,s}$; The PIC of URH itself is $PIC_{urh} = (C^{urh,p} - C^{urh,s}) \times 100\%/C^{urh,s}$; The Percentage of Deadline Violation (PDV) of DDS is defined as $PDV_{dds} = (f_{dds} - d) \times 100\%/d$. The PDV of URH is $PDV_{urh} = (f_{urh,p} - d) \times 100\%/d$.

The experimental results are analyzed by the multifactor analysis of variance (ANOVA) method [50]. The three main hypotheses (normality, homoskedasticity and independence of the residuals) are checked and accepted. In total, the deadlines of workflows are violated by the average percentage of 1% to 27% when URH is put into practice with stochastic task execution times. The average PICs of URH range from about 0% to 25%, which illustrates that the practical resource renting cost of URH is increased because of stochastic task execution times. The dynamic algorithm DDS get negative PDVs and lower (or equal) PICs than those of URH for most cases. This means that DDS can guarantee workflow deadlines and at the same time it gets lower resource renting costs than URH for most instances. Similarly, experimental results show that the runtime of DDS is usually within 10 milliseconds and is not longer than 100 milliseconds at most. Therefore, DDS is fast enough for the considered cloud BoT workflow applications.

The deadline factor, the pricing interval length, the maximum prediction deviation, the task execution time probability distribution type, delay threshold α and the estimation method of stochastic task execution times have a great impact on the performance of scheduling algorithms. Therefore, in the following we test all these aspects in detail.

5.4.1. Comparison under different deadline tightness

Figure 5 shows the means plot for PIC and PDV with 95.0% Tukey Honest Significant Difference (HSD) confidence intervals of URH and DDS algorithms with different deadline factors on different types of workflows. For Cybershake, Genomic and SIPHT workflows, experimental results show that DDS can guarantee the workflow deadline and obtain lower resource renting costs than URH. The reason is that bag-based delay strategy is helpful in decreasing the rental cost by considering a large number of tasks of a BoT together. The type and the number of VMs are determined by SIR dynamically according to BoT characteristics and task deadlines which can guarantee deadlines and decrease cost simutaneously. At the same time, SIR tries to minimize the total renting cost by selecting the VM with the cheapest estimated increased cost for each task greedily.

For Ligo and Montage workflows, deadlines are violated for most cases when URH is put into practice. On the contrary, DDS can guarantee deadlines with equal or a little higher resource renting cost. However, PICs of DDS for Ligo and Montage workflows with extremely tight deadlines are extremely larger than those of URH. The reason is as follows. The fastest workflow makespan D^s is calculated without considering VM setup times. Execution times of most Ligo and Montage tasks are within 100 seconds. When λ is small, the workflow deadline $\lambda \times D^s$ is very tight, which leads to extremely tight task deadlines for some tasks. New VMs are requested until tasks are ready which means that preparing VMs will consume much of the time of the task deadline. When the task deadline is extremely tight, a very short time or even no time of the task deadline is left for task processing. Therefore, a large number of the fastest VM type (usually the most expensive) is rented by DDS to finish the task as early as possible which increases the resource renting cost greatly. On the contrary, URH only schedules tasks according to the static schedule without considering deadline violations incurred by stochastic task execution times. Figure 6 shows an example of URH (left one) and DDS (right one) schedule results on ElasticSim with the same tight deadlines (1367 seconds) and the same task execution time sampling results. Each tab of ElasticSim shows VM provisioning and scheduling results of an algorithm. Each row of the tab displays states of a VM. The pale blue background rectangle of each row represents the rented time intervals of the VM. The time points of left and right sides of the rectangle are the start renting time and releasing time of the VM respectively. The text on the left of each background rectangle is the unique identification of the VM. Each vertical line on the background rectangle denotes the finish time of a rented interval. Foreground colour rectangles on background rectangles are assigned tasks. The time points of left and right sides of a colour rectangle are the start time and finish time of the corresponding task respectively. According to Figure 6, URH rented 13 $c4.2xLe$ VMs which got a finish time of 1407.6 seconds and a renting cost of 7.13 \$. DDS rented 11 $c4.4xL$ VMs which can grantee the workflow deadline with a finish time of 923 seconds but with a higher renting cost of 10.475 \$. In other words, DDS rented more expensive VM types to fulfill task deadlines while consumed additional cost.

5.4.2. Comparison under different pricing interval lengths

Figure 7 shows the means plots for PIC and PDV with 95.0% Tukey Honest Significant Difference (HSD) confidence intervals with different length of pricing intervals. Experimental results show that DDS algorithms still obtains equal or lower PICs than URH for most cases and can guarantee the deadlines. At the same time, the average PIC increases as the length of pricing interval increases from a general view. For example, PICs of cases with L=60 seconds are smaller than those of cases with longer pricing intervals. This is because a shorter pricing interval length is beneficial for avoiding the waste of rented VM intervals. The PDVs of most workflow types are stable for different lengths of pricing intervals. In summary, the length of pricing intervals has a little impact on the workflow finish time.

5.4.3. Comparison under different maximum prediction deviations

Figure 8, **??**, **??**, **??** and **??** show the means plots for PIC and PDV with 95.0% Tukey Honest Significant Difference (HSD) confidence intervals with different maximum prediction deviation of task execution times. In total, the PICs of DDS algorithms are lower than those of URH on Cybershake, Genomic and Sipht workflows. However, the PICs of DDS algorithms are larger than those of URH on Ligo and Montage workflows. The reason is again the presence of tight deadlines. The PDVs of all types of workflows are negative, which illustrates that workflow deadlines are satisfied by DDS for all degrees of prediction deviation. At the same time, the PDVs of most types of workflows are more stable than those of URH while the PDVs of URH increases as the maximum prediction deviation increases. This means that DDS algorithms are more robust for stochastic task execution times than URH. Not only the workflow deadlines are satisfied by DDS algorithms, but also the resource renting costs are decreased for most instances with diverse degree of maximum prediction deviations when compared to URH.

5.4.4. Comparison under different distribution types

Because results of uniform distribution have similar trend with the normal distribution on the above factors, some results of uniform distribution have been omitted which can be found in the attachment [51]. We just compare the two distribution types in total. Figure 9 shows the means plot for PIC and PDV with 95.0% Tukey Honest Significant Difference (HSD) confidence intervals of DDS with different task execution distribution types. Experimental results show that the uniform distribution type of task execution times has larger PICs than the normal distribution type in general, which means that the uniform distribution type has a larger impact on the resource renting cost. The reason is that variance of the uniform distribution type is larger than the gaussian distribution type, the DDS tends to rent more number of VM intervals and the assigned tasks are more likely to exceed planned VM intervals. The uniform distribution gets smaller PDVs than the normal distribution. The reason is that the standard deviation of the uniform distribution is larger than that of normal distribution. Because DDS algorithms make resource renting plan based on the sum of mean and standard deviation of stochastic task execution times, the estimated task execution time of uniform distribution is longer than that of normal distribution. Longer estimated task execution times will make DDS algorithms rent more number of parallel VMs or faster VMs.

5.4.5. Comparison under different delay threshold α

According to Figure 5, 7 and 8(Result Set 1), DDSRR30 gets lower PICs than DDSRR0, DDSRR5 and DDSRR100. It means that DDSRR30 with $\alpha = 30$ gets a better tradeoff between the using of global information by scheduling a batch of tasks together and the length of left task execution times.

5.4.6. Comparison with DDS_MAX

In this paper, SIR tries to fulfill task deadlines by using the sum of task execution time expectation and standard deviation as an approximation of the practical task execution time. We also evaluated the performance of DDS when the maximum value of the stochastic task execution time is used. According to the Result Set 1, DDS_MAX gets higher PICs and lower PDVs than DDSRR100. It means that DDS_MAX can decrease makespans further but with higher VM renting cost. The reason is that the maximum value of the task execution time over-estimates the actual task execution time which makes the SIR rent more number of VM intervals or fast VMs. The result is consistent with the conclusion of Tang et al. [24].

5.5. Comparison with MOHEFT

For each tested workflow instance, MOHEFT obtain a set of schedule which were tradeoffs between makespan and the resource renting cost. For each workflow instance, MOHEFT obtains a set of Pareto schedule $\tau = \{p_k^{mo}\}, k = 1, 2, 3..., K$, in which $p_k = (\mathcal{T}_k^{mo,s}, C_k^{mo,s})$ is a schedule with execution time $\mathcal{T}_k^{mo,s}$ and renting cost $C_k^{mo,s}$ (in this paper, $K = 20$). A larger value of the parameter K (size of the remaining solution after cutting each iteration) for MOHEFT generates better effectiveness and a longer computation time. When the K of MOHEFT is infinite, MOHEFT leads to an exhaustive search which is extremely time consuming. When the URH and MOHEFT is used in the ElasticSim, the practical resource renting cost and finish time of URH and MOHEFT are different from those of original static schedule. When a static schedule p_k^{mo} of MOHEFT is put into practice with stochastic task execution times, the practical performance is labeled by $(\mathcal{T}_k^{mo,d},C_k^{mo,d})$. For fair comparison, the execution time $\mathcal{T}_k^{mo,s}$ of each MOHEFT solution $p_k \in \tau$ is adopted as the deadline of URH and DDS algorithms. For each given deadline $\mathcal{T}_k^{mo,s}$ in τ, the URH first got a static schedule p_k^{urh} with performance $(\mathcal{T}_k^{mo,s},C_k^{urh,s})$. When p_k^{urh} is put into practice, the practical performance is $(\mathcal{T}_k^{urh,d},C_k^{urh,d})$. For each given deadline $\mathcal{T}_k^{mo,s}$, the DDS gets the practical performance $(\mathcal{T}_k^{dds,d},C_k^{dds,d})$ directly. In total, K schedules are generated by URH or DDS for each workflow instance. The performance of DDS is compared with the practical performances of MOHEFT and URH. Let PIC and PDV of URH be $PIC_k^{mo} = (C_k^{mo,d} - C_k^{urh,s}) \times 100\%/C_k^{urh,s}$ and $PDV_k^{mo} = (\mathcal{T}_k^{mo,d} - \mathcal{T}_k^{mo,s}) \times 100\%/\mathcal{T}_k^{mo,s}$. $PIC_k^{dds} = (C_k^{dds,d} - C_k^{urh,s}) \times 100\%/C_k^{urh,s}$ and $PDV_k^{dds} = (\mathcal{T}_k^{dds,d} - \mathcal{T}_k^{mo,s}) \times 100\%/\mathcal{T}_k^{mo,s}$ represent PIC and PDV of a DDS algorithm respectively.

According to experimental results, the computation time of MOHEFT rises to about 650 seconds for workflows with 1000 tasks as K increased to 20. The total execution time of DDS algorithms on the instances with 1000 tasks is only about 2.5 seconds which is the sum of the DDS computation time and the simulation time. 650s is significantly longer than K times of 2.5s. We aim to prove that DDS algorithms still generat lower resource renting costs than MOHEFT for the same deadline.

Figure 10 (Result Set 2) show the means plots for PIC and PDV with 95.0% Tukey Honest Significant Difference (HSD) confidence intervals of URH, DDS and MOHEFT with different Deadlinefactors (the horizontal axis represents 5 times of Deadlinefactors). Experimental results show that DDSRR30 gets lower PICs than MOHEFT for LIGO, Montage and SIPHT workflows. For the Cybershake and Genomic workflows, on the contrary, DDSRR30 only gets lower PICs when Deadlinefactors is smaller than 10 and 50 respectively. According to the density functions shown in Figure 11, most Deadlinefactors generated from solutions of MOHEFT are smaller than 10 and 50 respectively for the two types of workflows. In other words, DDSRR30 gets better performance than MOHEFT for most cases. PICs (and PDVs) of URH and DDSRR30 in Result Set 2 have the same trend with those of URH and DDSRR30 in Result Set 1. At the same time, DDSRR30 can guarantee deadlines, while deadlines are violated by MOHEFT for most cases.

6. Conclusions and future research

In this paper, a dynamic cloud resource provisioning and scheduling scheduling algorithm DDS is proposed to minimize the resource renting cost while meeting workflow deadlines. New VMs are rented by the DDS dynamically according to the practical execution state and the estimated task execution times to fulfill the workflow deadline. The bag-based deadline division and bag-based delay scheduling strategies consider each BoT as a whole to decrease the total renting cost. The single-type based VM interval renting strategy aims to minimize the resource renting cost while improving the algorithm efficiency. Experimental results show that the DDS can guarantee the workflow deadline for all instances and obtains lower resource renting costs than URH on most instances. However, the resource renting cost are not decreased as much compared with the static algorithm URH. In particularly, DDS gets worse performance than URH on some types of workflows with extremely tight deadlines. This is because the expectation-and-variance based task execution time estimation methods overestimate the practical task execution times to some degree. Therefore, developing more appropriate task execution time estimation methods and deadline division strategies considering VM setup times for workflows with extremely tight deadlines is a promising future line of research.

Acknowledgements

This work is supported by the National Natural Science Foundation of China (Grant No.61602243), the Natural Science Foundation of Jiangsu Province (Grant No.BK20160846), Jiangsu Key Laboratory of Image and Video Understanding for Social Safety (Nanjing University of Science and Technology, Grant No. 30916014107), the Fundamental Research Funds for the Central University (Grant No.30916015104). Rubén Ruiz is partially supported by the Spanish Ministry of Economy and Competitiveness, under the project "SCHEYARD" (No.DP12015-65895-E) co-financed by FEDER funds

References

[1] J. O. Gutierrez-Garcia, K. M. Sim, Agent-based cloud bag-of-tasks execution, Journal of Systems and Software 104 (2015) 17–31.

[2] X. Fei, S. Lu, C. Lin, A MapReduce-enabled scientific workflow composition framework, in: IEEE International Conference on Web Services, 2009, pp. 663–670.

[3] Y. Wang, W. Shi, Budget-driven scheduling algorithms for batches of MapReduce jobs in heterogeneous clouds, IEEE Transactions on Cloud Computing 2 (3) (2014) 306–319.

[4] S. Tao, H. Feng, Z. Xu, Q. Li, Image degradation and recovery based on multiple scattering in remote sensing and bad weather condition, Optics Express 20 (15) (2012) 16584–16595.

[5] R. Bajaj, D. P. Agrawal, Improving scheduling of tasks in a heterogeneous environment, IEEE Transactions on Parallel and Distributed Systems 15 (2) (2004) 107–118.

[6] R. H. Möhring, Minimizing costs of resource requirements in project networks subject to a fixed completion time, Operations Research 32 (1) (1984) 89–120.

[7] E. Demeulemeester, B. D. Reyck, B. Foubert, W. Herroelen, M. Vanhoucke, New computational results on the discrete time/cost trade-off problem in project networks, Journal of the Operational Research Society 49 (11) (1998) 1153–1163.

[8] P. De, G. Jb., W. Ce., D. Ej., Complexity of the discrete timeccost tradeoff problem for project networks, Operations Research 45 (2) (1997) 302–306.

[9] G. Galante, L. C. Erpen De Bona, A. R. Mury, B. Schulze, R. da Rosa Righi, An analysis of public clouds elasticity in the execution of scientific applications: a survey, Journal of Grid Computing 14 (2) (2016) 193–216.

[10] X. Zuo, G. Zhang, W. Tan, Self-adaptive learning PSO-based deadline constrained task scheduling for hybrid IaaS cloud, IEEE Transactions on Automation Science and Engineering 11 (2) (2014) 564–573.

[11] R. V. D. Bossche, K. Vanmechelen, J. Broeckhove, Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds, Future Generation Computer Systems 29 (4) (2013) 973–985.

[12] I. A. Moschakis, H. D. Karatza, Multi-criteria scheduling of bag-of-tasks applications on heterogeneous interlinked clouds with simulated annealing, Journal of Systems and Software 101 (C) (2015) 1–14.

[13] X. Li, Z. Cai, Elastic resource provisioning for cloud workflow applications, IEEE Transactions on Automation Science and Engineering, In press, DOI:10.1109/TASE.2015.2500574 (2016).

[14] E. K. Byun, Y. S. Kee, J. S. Kim, S. Maeng, Cost optimized provisioning of elastic resources for application workflows, Future Generation Computer Systems 27 (8) (2011) 1011–1026.

[15] W. Chen, R. F. D. Silva, E. Deelman, R. Sakellariou, Using imbalance metrics to optimize task clustering in scientific workflow executions, Future Generation Computer Systems 46 (C) (2014) 69–84.

[16] S. Abrishami, M. Naghibzadeh, D. Epema, Deadline-constrained workflow scheduling algorithms for IaaS clouds, Future Generation Computer Systems 29 (1) (2013) 158–169.

[17] J. J. Durillo, R. Prodan, Multi-objective workflow scheduling in Amazon EC2, Cluster Computing 17 (2) (2013) 169–189.

[18] Z. Cai, X. Li, J. N. D. Gupta, Heuristics for provisioning services to workflows in XaaS clouds, IEEE Transactions on Services Computing 9 (2) (2016) 250–263.

[19] A. Lastovetsky, J. Twamley, Towards a realistic performance model for networks of heterogeneous computers, in: International Federation for Information Processing Digital Library; High PERFORMANCE Computational Science and Engineering, 2004, pp. 39–57.

[20] M. Malawski, G. Juve, E. Deelman, J. Nabrzyski, Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds, Future Generation Computer Systems 48 (2015) 1–18.

[21] W. Zheng, R. Sakellariou, Stochastic DAG scheduling using a Monte Carlo approach, Journal of Parallel and Distributed Computing 73 (12) (2013) 1673–1689.

[22] M. Skutella, M. Uetz, Stochastic machine scheduling with precedence constraints, SIAM Journal on Computing 34 (4) (2005) 788–802.

[23] A. Kamthe, S.-Y. Lee, A stochastic approach to estimating earliest start times of nodes for scheduling DAGs on heterogeneous distributed computing systems, Cluster Computing 14 (4) (2005) 377–395.

[24] X. Tang, K. Li, G. Liao, K. Fang, F. Wu, A stochastic scheduling algorithm for precedence constrained tasks on grid, Future Generation Computer Systems 27 (8) (2011) 1083–1091.

[25] R. Sandhu, S. K. Sood, Scheduling of big data applications on distributed cloud based on QoS parameters, Cluster Computing 18 (2) (2014) 817–828.

[26] E.-K. Byun, Y.-S. Kee, J.-S. Kim, E. Deelman, S. Maeng, BTS: Resource capacity estimate for time-targeted science workflows, Journal of Parallel and Distributed Computing 71 (6) (2011) 848–862.

[27] M. Shojafar, S. Javanmardi, S. Abolfazli, N. Cordeschi, Fuge: A joint metaheuristic approach to cloud job scheduling algorithm using fuzzy theory and a genetic method, Cluster Computing 18 (2) (2015) 829–844.

[28] T. Kaur, I. Chana, Energy aware scheduling of deadline-constrained tasks in cloud computing, Cluster Computing, In press, DOI:10.1007/s10586-016-0566-9 (2016).

[29] N. Akhter, M. Othman, Energy aware resource allocation of cloud data center: review and open issues, Cluster Computing, In press, DOI:10.1007/s10586-016-0579-4 (2016).

[30] J. Shi, J. Luo, F. Dong, J. Zhang, J. Zhang, Elastic resource provisioning for scientific workflow scheduling in cloud under budget and deadline constraints, Cluster Computing 19 (1) (2016) 167–182.

[31] J. Liu, E. Pacitti, P. Valduriez, M. Mattoso, A survey of data-intensive scientific workflow management, Journal of Grid Computing 13 (4) (2015) 457–493.

[32] A. Verma, S. Kaushal, Cost-time efficient scheduling plan for executing workflows in the cloud, Journal of Grid Computing 13 (4) (2015) 495–506.

[33] S. Su, J. Li, Q. Huang, X. Huang, K. Shuang, J. Wang, Cost-efficient task scheduling for executing large programs in the cloud, Parallel Computing 39 (4C5) (2013) 177–188.

[34] M. A. Iverson, F. Ozguner, L. C. Potter, Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment, IEEE Transactions on Computers 48 (12) (1999) 1374–1379.

[35] L. David, I. Puaut, Static determination of probabilistic execution times, in: Euromicro Conference on Real-Time Systems, 2004, pp. 223–230.

[36] R. Duan, F. Nadeem, J. Wang, Y. Zhang, A hybrid intelligent method for performance modeling and prediction of workflow activities in grids, in: 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009, pp. 339–347.

[37] L. Carrington, A. Snavely, N. Wolter, A performance prediction framework for scientific applications, Future Generation Computer Systems 22 (3) (2006) 336–346.

[38] S. Singh, I. Chana, A survey on resource scheduling in cloud computing: Issues and challenges, Journal of Grid Computing 14 (2) (2016) 217–264.

[39] M. A. Rodriguez, R. Buyya, Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds, IEEE Transactions on Cloud Computing 2 (2) (2014) 222–235.

[40] Z. Tang, M. Liu, A. Ammar, K. Li, K. Li, An optimized MapReduce workflow scheduling algorithm for heterogeneous computing, Journal of Supercomputing 72 (6) (2014) 1–21.

[41] Z. Cai, X. Li, R. Ruiz, Cloud YARN resource provisioning for task-batch based workflows with deadlines, Technical report `https://github.com/czcnjust/ElasticSim/blob/master/Technicalreport201500805.pdf`.

[42] R. L. Graham, E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Annals of Discrete Mathematics 5 (4) (1979) 287–326.

[43] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, R. Buyya, Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Software: Practice and Experience 41 (1) (2011) 23–50.

[44] A cloud workflow simulator called ElasticSim, `https://github.com/czcnjust/ElasticSim/wiki`, accessed, 2016.6.30.

[45] A workflow generator, `https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator`, accessed, 2016.6.30.

[46] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, K. Vahi, Characterization of scientific workflows, in: Third Workshop on Workflows in Support of Large-Scale Science, IEEE, 2008, pp. 1–10.

[47] L. Epstein, J. Sgall, Approximation schemes for scheduling on uniformly related and identical parallel machines, Algorithmica 39 (1) (2004) 43–57.

[48] C. Rutten, D. Recalde, P. Schuurman, T. Vredeveld, Performance guarantees of jump neighborhoods on restricted related parallel machines , Operations Research Letters 40 (4) (2012) 287–291.

[49] H. Topcuoglu, S. Hariri, M. Y. Wu, Performance-effective and low-complexity task scheduling forheterogeneous computing, IEEE Transactions on Parallel and Distributed Systems 13 (3) (2002) 260–274.

[50] T. Bartz-Beielstein, M. Chiarandini, L. Paquete, M. Preuss, Experimental methods for the analysis of optimization algorithms, Springer, 2010.

[51] Results of dds and urh algorithms under the uniform distribution type, `https://github.com/czcnjust/ElasticSim/blob/master/FGCS-ZhichengCai%20-attach.pdf`, accessed, 2016.6.30.

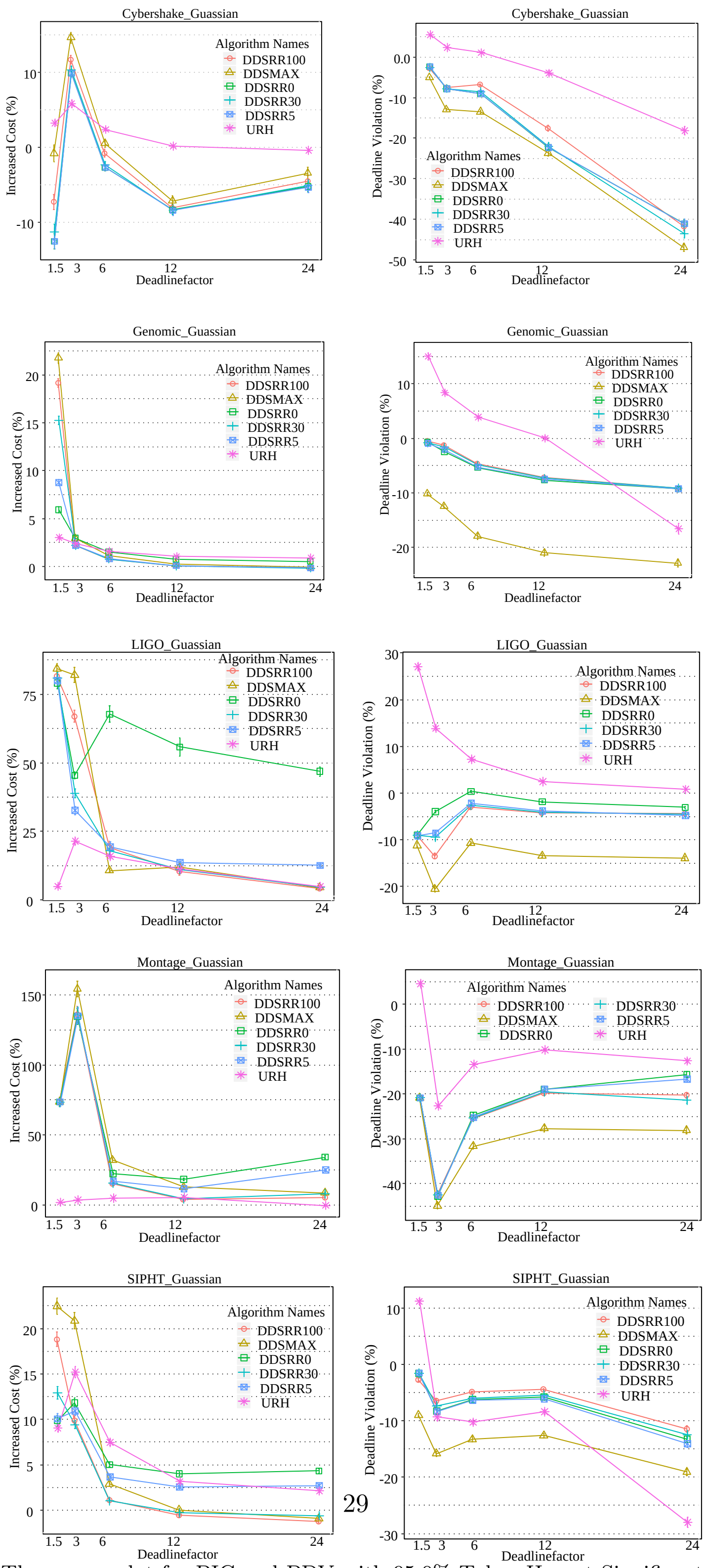


Figure 5: The means plot for PIC and PDV with 95.0% Tukey Honest Significant Difference (HSD) confidence intervals of URH and DDS algorithms with different deadline factors

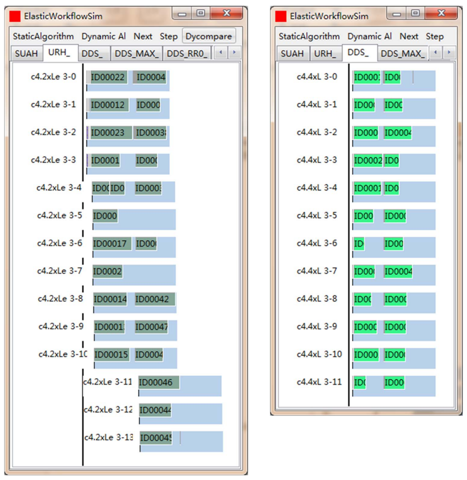


Figure 6: Schedule results of URH and DDS with tight deadlines

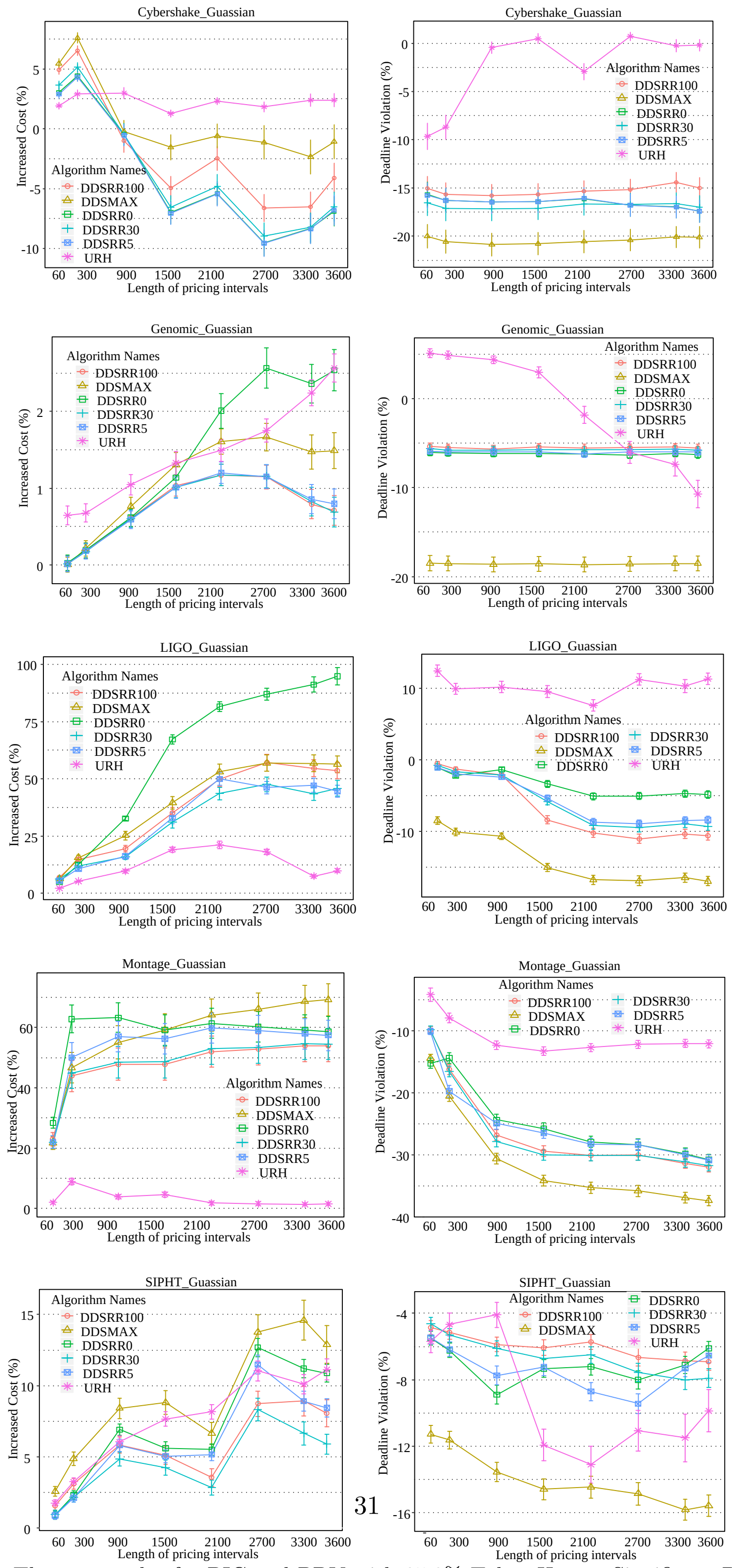


Figure 7: The means plot for PIC and PDV with 95.0% Tukey Honest Significant Difference (HSD) confidence intervals of URH and DDS with different length of pricing intervals

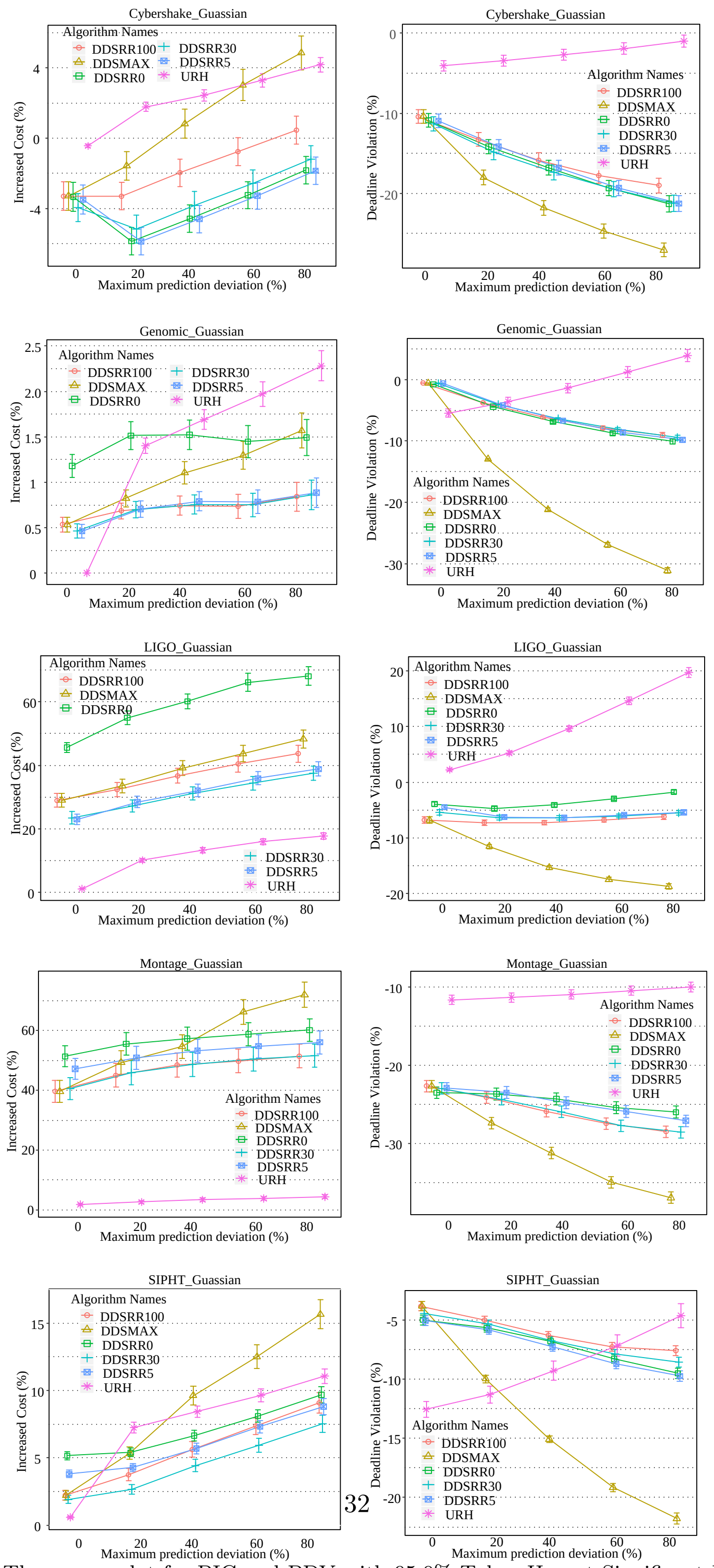
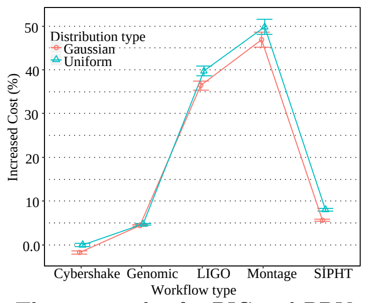


Figure 8: The means plot for PIC and PDV with 95.0% Tukey Honest Significant Difference (HSD) confidence intervals of URH and DDS with different maximum prediction deviation of task execution times



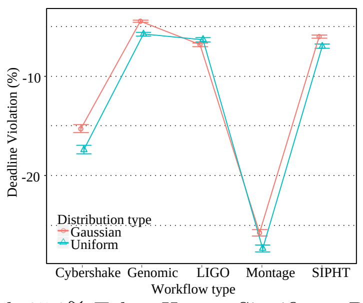


Figure 9: The means plot for PIC and PDV with 95.0% Tukey Honest Significant Difference (HSD) confidence intervals of DDS with different distribution types and workflow types

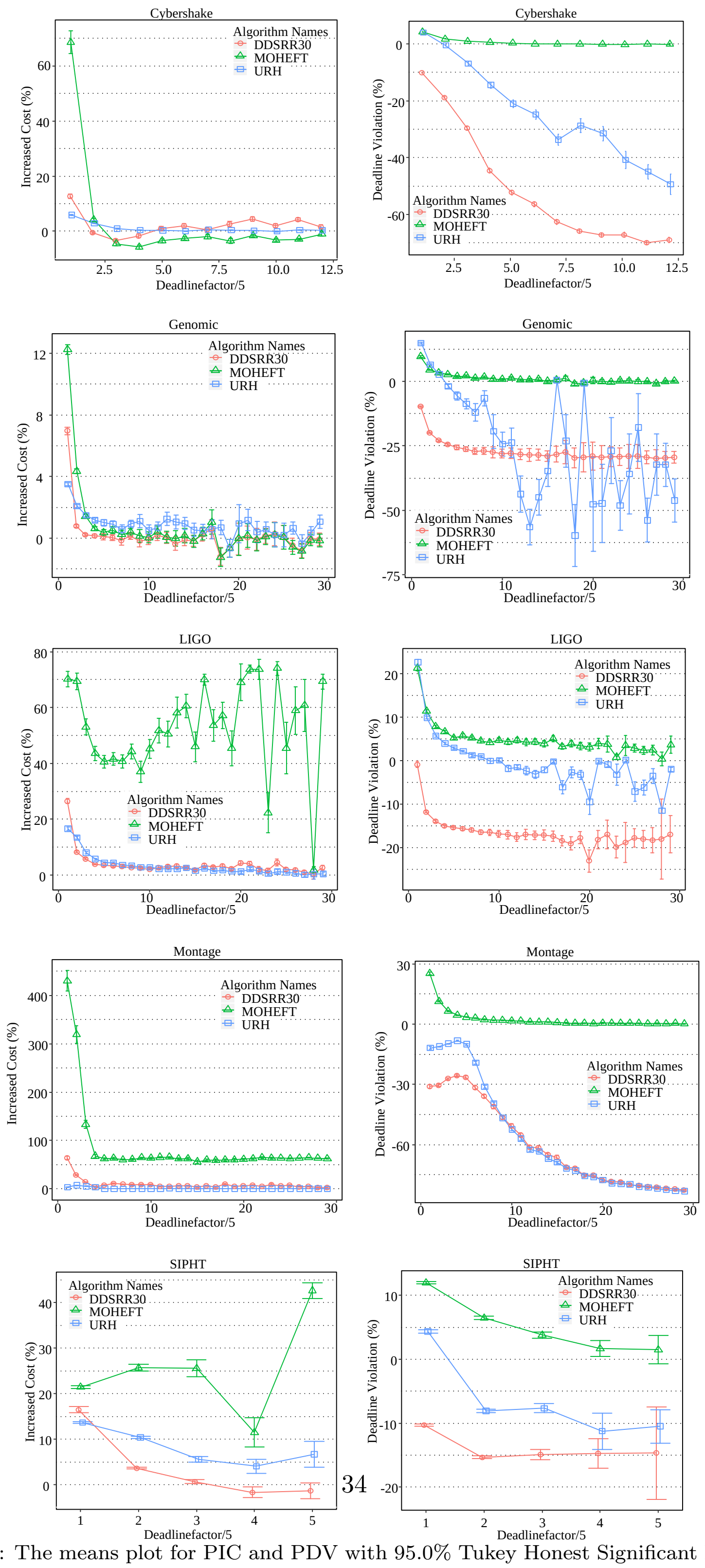


Figure 10: The means plot for PIC and PDV with 95.0% Tukey Honest Significant Difference (HSD) confidence intervals of URH, DDS and MOHEFT on Cybershke workflows

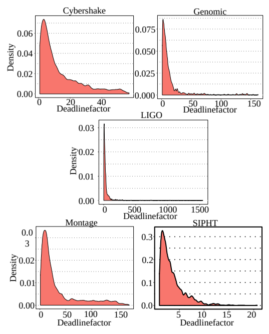


Figure 11: Density of Deadlinefactor of MOHEFT results