Chen, L.; Li, X.; Ruiz García, R. (12-2). Idle block based methods for cloud workflow scheduling with preemptive and non-preemptive tasks. Future Generation Computer Systems. 89:659-669. https://doi.org/10.1016/j.future.2018.07.037

Idle block based methods for cloud workflow scheduling with preemptive and non-preemptive tasks

Long Chen[a], Xiaoping Li[a,*], Rubén Ruiz[b]

[a] *School of Computer Science and Engineering, Southeast University, Nanjing, China*
[b] *Universitat Politècnica de València, València, Spain*

Abstract

Complex workflow applications are widely used in scientific computing and economic analysis, which commonly include both preemptive and non-preemptive tasks. Cloud computing provides a convenient way for users to access different resources based on the "pay-as-you-go" model. However, different resource renting manners (reserved, on-demand or spot) are usually provided by the service provider. The spot instances provide a dynamic and cheaper manner comparing to the on-demand one. However, failures often occur due to the fluctuations of the price of the instance. It is a big challenge to determine the appropriate amount of spot and on-demand resources for workflow applications with both preemptive and non-preemptive tasks. In this paper, the workflow scheduling problem with both spot and on-demand instances is considered. The objective is to minimize the total renting cost under deadline constrains. An idle time block-based method is proposed for the considered problem. Different idle time block-based searing and improving strategies are developed to construct schedules for workflow applications. Schedules are improved by a forward and backward moving mechanism. Experimental and statistical results demonstrate the effectiveness of the proposed algorithm over a lot of tests with different sizes.

Keywords: Resource Allocation; Hybrid Resources Renting; On-demand instances; Cloud Computing

*Corresponding author
Email addresses: longc@seu.edu.cn (Long Chen), xpli@seu.edu.cn (Xiaoping Li), rruiz@eio.upv.es (Rubén Ruiz)

1. Introduction

Cloud computing is a new market-oriented resource provisioning and sharing paradigm, which enable users to access resources from anywhere and at anytime. In cloud computing, resources (server, network, storage, platform, software, etc.) are virtualized as services. Users need not to possess and manage large amounts of services, users just rent and pay the service providers when they require such services. These paradigms spare users from expensive purchases effectively and no less expensive run and maintenance costs. Cloud computing makes it convenient for users to access resources from anywhere based on the pay-as-you-go model which reduces the cost significantly. Therefore, allocating appropriate resources to users is one of the crucial problems in cloud computing.

In cloud computing, workflow applications are commonly used to represent users' requests, which could describe a wide range of complex scientific and economic applications [1], e.g., astronomy application (Montage), bio-informatics project (SIPHT) and astrophysical application (LIGO). The workflow application commonly contains two types of tasks: non-preemptive tasks and preemptive tasks. Non-preemptive tasks are continuous tasks and cannot be terminated during their execution, such as batch processing, encoding and rendering, and continuous integration. If the resources of non-preemptive tasks are crashed, these tasks must be redone from the beginning. The preemptive tasks are interruptible tasks and can be terminated during their execution, e.g., web crawler application. If the resources of preemptive tasks are crashed, these tasks must be reallocated to other resources just from the crashed point.

Three types of resource renting manners are usually provided by the service providers, e.g., Amazon EC2 [2] provides three type of instances: reserved, on-demand or spot. Google Compute Engine cloud has also announced the preemptive virtual machines as the supplements of common reserved and on-demand virtual machines. The reserved manner enables users to pay for required resources at one-time for a long period. Thus, a significant discount could be received and the average cost is decreased. However, the resource utilization rate of reserved resources is usually lower as the resources could not be fully used during the renting period. The on-demand manner enables users to

rent and pay resources by hours according to their demands. Though the on-demand instance is expensive, the resource utilization of on-demand instances is usually high. The spot manner enables users to bid for the resource capacity. The price of spot instances fluctuates according to the resource supply and demand capacities. If the users' bid price is higher than the spot price, users' requests are fulfilled. Instances are kept by the current user until new higher prices are bid. Spot instances provide a dynamic and cheaper manner for renting resources from the cloud.

In this paper, the cloud workflow scheduling problem with both non-preemptive and preemptive tasks is considered. The objective is to minimize the total resource renting cost. As the scientific workflow application is a short term compute-intensive application, the long term reserved instances are not included. If only the on-demand instances are included, the high unit cost makes the total resource renting cost high. If the general spot instances are considered, the out-of-bid failure may occur and make it improper for the non-preemptive tasks (the redoing process will occupy more resources). Thus, the spot block instances are considered in this paper. The spot block instance is an extension of the spot instance, which will not be terminated with a specified duration of 1, 2, 3, 4, 5, or 6 hours [2]. The price of a spot block instance depends on the specified duration. When a request with a duration is fulfilled, the price for the spot block instance is fixed, and this price remains in effect until the instance terminates. The spot block instance runs until the task is terminated or the duration period ends. These advantages make them ideal for non-preemptive tasks that cannot be interrupted during the execution. If a non-preemptive task is allocated to a spot block instance, its execution time must be less than the spot block time. Otherwise, the on-demand instance must be rented to ensure the non-preemptive task will not be interrupted. A preemptive task can be executed on any possible instances. It can be interrupted and wait to be reallocated to other available instances.

The problem under study considers a new cloud workflow scheduling problem. Since workflow and DAG scheduling problems are well known NP-hard problem [3], it is natural that the much complex considered problem is NP-hard. Resource renting and task scheduling are the two main challenges for the problem. During the resource renting process, it is hard to determine the type and the renting manners (spot block and

the block period or on-demand) of instances. During the task scheduling process, it is hard to segment the preemptive tasks and find available idle blocks for both preemptive and non-preemptive tasks. The considered complex workflow scheduling problems are mathematically modeled. The idle time block-based method (ITB) is proposed for the problem under study which is composed of five components: deadline division (DD), sequences initialization (ASI), idle-time block searching (IBS), idle time block-based schedule construction (ISC) and schedule improvement (SI). Parameters and components of the proposed algorithm are calibrated and analyzed over a number of instances using the multi-factor analysis of variance technique. The proposal is compared with existing algorithms for similar problems.

The rest of the paper is organized as follows. Section 2 establishes the related works. The mathematical model are described in Section 3. The proposed ITB method is established in Section 4. Computational results are shown in Section 5, followed by conclusions in Section 6.

2. Related Works

Workflow scheduling is a hot topic in recent years. It can be supervised and executed in many distributed or cloud systems, such as Pegasus [4], Askalon [5], Google MapReduce [6] and Amazon EC2 [7].

In the traditional utility grids environment, resources were usually geographically dispersed and encapsulated as services and provided to the users. There are two main objectives: cost optimization under deadline constraints and execution time optimization under budget constraints [8]. Methods for time optimization include list scheduling algorithm [9], cluster based algorithm [10], duplication based algorithm [11], greedy randomized adaptive search [12] and ant colony optimization approach [13]. Common methods for cost optimization include the deadline-MDP algorithm [14], DET (Deadline Early Tree) algorithm [15], PCP (Partial Critical Paths) algorithm [16] and the CPI (Critical Path-based Iterative) heuristic [17]. Other related works on allocating resources to workflow applications include improving QoS in computational grids [18], workflow applications with security constraints [19] and workflow scheduling

with multiple objectives [20][21].

In cloud computing environment, resources were usually geographically concentrated, which were virtulized as a virtual machine and provided to the users. Different resource renting manners were adopted by different papers. Chen et al. [22] proposed a Precedence Tree based Heuristic (PTH) for the long period periodical workflow scheduling with reserved resources. In a follow up work [23], they developed an Adaptive Probabilistic Algorithm (APA) for cloud workflow scheduling with both reserved and on-demand resources. A Balanced Time Scheduling (BTS) algorithm was proposed by Byun et al. [24] for scheduling homogeneous resources to a workflow with deadline constrained. In a follow up work of Byun et al. [25], a Partitioned Balanced Time Scheduling (PBTS) algorithm was proposed for scheduling the On-demand homogeneous resources to workflow applications. PBTS considers time partitions in the algorithm and minimizes the amount of resources for each time partition. Abrishami et al. [16] proposed a QoS-based Partial Critical Paths (PCP) workflow scheduling algorithm on utility Grids, and the PCP algorithm was modified for On-demand cases [26] in Cloud computing. Two algorithms, IC-PCP and IC-PCPD2, were proposed which are different from PCP in three aspects: On-demand resource provisioning, homogeneous networks, and the pay-as-you-go pricing model. Cai et al. [27] divided the workflow scheduling problem with On-demand resource provisioning into two sub-problems: service mapping and task tabling on sharable resources. Two heuristics CPIS (Critical Path based Iterative heuristic with Shortest services) and LHCM (List based Heuristic considering Cost minimization and Mach degree maximization) were developed for the sub-problems, respectively.

There are only a few studies on workflow scheduling with spot instances. Poola et al. [28] proposed a fault tolerant algorithm which schedules tasks on Cloud with spot and on-demand instances. The objective was to reduce the total renting cost with the workflow deadline constrained. In their following up work [29], an adaptive just-in-time scheduling algorithm was proposed for scientific workflows. This algorithm used both spot and on-demand instances to reduce cost and provide fault tolerance. Jung et al. [30] considered task balanced workflow scheduling scheme to reduce the out-of-bid situation and the total task completion time. In another work of Jung et al. [31], a

Genetic Algorithm (GA)-based workflow scheduling scheme was proposed to find the optimal task size in a spot instance-based cloud environment without increasing users' budgets.

To the best of our knowledge, there is no existing work considering workflow scheduling both non-preemptive and preemptive tasks. However, these two types of tasks are commonly exists in scientific workflows. The short-term on-demand manner is usually considered for workflow scheduling in the existing literature. A few studies have also considered the spot manner. However, the high unit cost and out-of-bid failure make these two resource renting manner more expensive. Therefore, scheduling workflow applications with both non-preemptive and preemptive tasks on on-demand and spot block instances is considered in this paper.

3. Problem Description and Formulation

3.1. Framework of the Problem

Figure 1 shows the framework of cloud workflow scheduling with preemptive and non-preemptive tasks. Users send their requests to the cloud service providers (CSP). The CSP mainly includes three modules: workflow scheduling module, resource renting module and the virtual cloud center. The workflow scheduling module decomposes users' workflow applications into a set of non-preemptive and preemptive tasks and submit the tasks to the resource renting module. According to the tasks, the resources renting module rent proper virtual machine instances from the virtual cloud center. The workflow scheduling module allocate the tasks to each available rented instance. During the whole process, the Qos (deadlines, start times, etc.) of each user must be satisfied.

3.2. Mathematical model

The workflow application is represented by a task-on-node Directed Acyclic Graph (DAG) $G(V, E)$, in which $V = \{v_0, \ldots, v_n\}$ are the tasks and $E = \{(v_i, v_j)|v_i \in V, v_j \in V, i < j\}$ are the edges in G. Each edge $E = \{(v_i, v_j)\}$ denotes the dependencies between task v_i and v_i. Two dummy nodes v_0 and v_n are denoted, which are

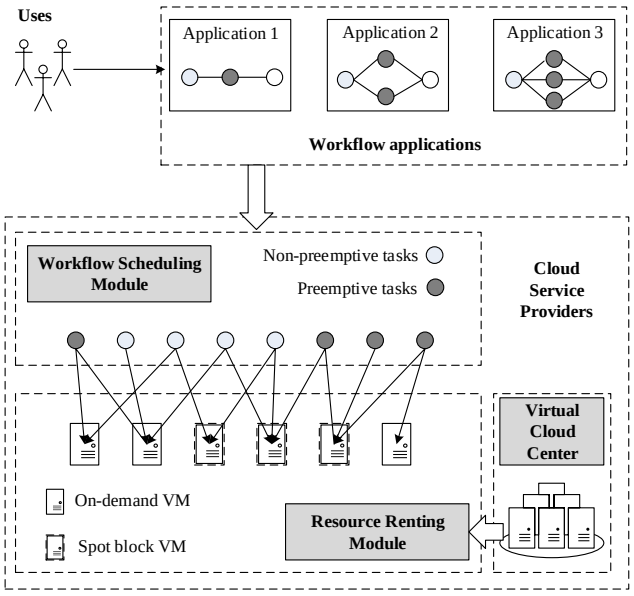

Figure 1: Cloud workflow scheduling with preemptive and non-preemptive tasks.

used to describe the dummy start and end tasks of the workflow. The two task sets $\mathbb{M}$ and $\mathbb{R}$ are used to denote the preemptive and non-preemptive tasks, respectively. The task v_i can be processed on one or several rented instances. Only homogeneous virtual machine instances are considered in this paper (virtual machines with the same capacity). Different physical machines are virtualized to the virtual machine with the same configuration. Each task can be executed on each idle virtual machine instance. Let the processing time of task v_i on a single virtual machine be P_i. For a preemptive task $v_i \in \mathbb{M}$, the task can be separated and allocated on the currently available virtual machines during the scheduling process. However, for a non-preemptive task $v_i \in \mathbb{R}$, it can only be processed on a single on-demand or spot block instance. Let C^o be the unit cost of the on-demand instance, $C^s_{tk}, k \in \{1, 2, 3, 4, 5, 6\}$ be the unit cost of the spot block instance at time t with the block time k. The deadline of the entire workflow is given by D. During the scheduling process, Let s_i and f_i be the start and finish times of task v_i, the total number of rented resources be H.

Then, the objective of the considered problem is denoted as follows. $C^o \times y_h \times T_h$

is the cost of on-demand virtual machines. $C^s_{E_h T_h} \times (1 - y_h) \times T_h$ calculates the cost of spot block virtual machines, in which E_h is the start time of a spot block instance and T_h is the block time.

$$\min \left\{ \sum_{h=1}^{H} (C^o \times y_h \times T_h + C^s_{E_h T_h} \times (1 - y_h) \times T_h) \right\} \tag{1}$$

Two types of binary variables are denoted. The binary variable x_{iht} in Equation (2) takes 1 if and only if task v_i is executed on VM h at time t. The binary variable y_h in Equation (3) takes 1 if and only if the VM h is an on-demand instance and takes 0 if and only if the VM h is an spot instance.

$$x_{iht} = \begin{cases} 1 \;\; v_i \text{ is processed on VM } h \text{ at time } t, \; \forall i \in \{0, \\ \quad \ldots, n\}, \; \forall h \in \{1, \ldots, H\}, \; \forall t \in \{0, \ldots, D\} \\ 0 \;\; \text{others} \end{cases} \tag{2}$$

$$y_h = \begin{cases} 1 \;\; h \text{ is on-demand instance}, \; \forall h \in \{1, \ldots, H\} \\ 0 \;\; h \text{ is spot instance}, \; \forall h \in \{1, \ldots, H\} \end{cases} \tag{3}$$

$$T_h = \sum_{i=0}^{n} \sum_{t=0}^{D} x_{iht}, \; \forall h \in \{1, \ldots, H\} \tag{4}$$

$$E_h = \sum_{i=0}^{n} \sum_{t=0}^{D} x_{iht} \times t, \; \forall h \in \{1, \ldots, H\} \tag{5}$$

$$s_i = \min_{t \in \{0, \ldots, D\}} (\sum_{h=1}^{H} t \times x_{iht}), \; \forall i \in \{0, \ldots, n\} \tag{6}$$

$$f_i = \max_{t \in \{0, \ldots, D\}} (\sum_{h=1}^{H} t \times x_{iht}), \; \forall i \in \{0, \ldots, n\} \tag{7}$$

$$m_i = \begin{cases} \sum_{h=1}^{H} \sum_{t=0}^{D} x_{iht}, & \forall v_i \in \mathbb{M} \\ 1, & \forall v_i \in \mathbb{R} \end{cases} \tag{8}$$

$$p_i = \lceil P_i / m_i \rceil \tag{9}$$

$$s_i + p_i \leqslant s_j, \; \forall (i, j) \in E \tag{10}$$

$$f_n \leqslant D \tag{11}$$

$$\sum_{h=1}^{H} x_{iht} = 1,\ \forall t \in \{0, \ldots, D\},\ \forall v_i \in \mathbb{R} \tag{12}$$

$$\sum_{h=1}^{H} \sum_{i=0}^{n} x_{iht} = 1,\ \forall t \in \{0, \ldots, D\} \tag{13}$$

The renting period of on-demand instances and spot block instances are calculated by Equation (4) The start time of the spot block instance are calculated in Equation (5). (6) and (7) calculate the start time and the finish time of task v_i respectively. Equation (8) calculates the virtual machines instances required for a task v_i. Equation (9) calculates the related processing time of a task v_i. Precedence and deadline constraints of the workflow are specified by formula (10) and (11). Equation (12) ensures that all the non-preemptive task can only be executed on one single virtual machine during the whole scheduling process. Equations (13) ensures each virtual machine only execute one task at a time.

4. Idle time block-based method

Rule-based heuristics are common methods for workflow scheduling problems [9]. An idle time block-based method (ITB) is proposed for the problem under study in this paper. ITB is composed of five components: allocation sequences initialization (ASI), deadline division (DD), idle-time block searching (IBS), task and block mapping (TBM), schedule improvement (SI). In the procedure DD, workflow deadlines are divided into task deadlines to balance the idle time block between tasks. In the SI, an initial task allocation sequence is determined according to the priorities of each task. In the IBS, idle-time blocks are searched and calculated. The procedure ISC schedules tasks sequentially based on the idle time block. The schedule is improved by adjusting preemptive tasks using the SI procedure. Details of the idle time block-based method is shown in Algorithm 1

4.1. Allocation sequences initialization (ASI)

The idle time block-based method (ITB) is a list based workflow scheduling algorithm [9]. First, The allocation sequences of tasks are determined by the ASI procedure.

Algorithm 1: Idle Time Block-based method (ITB)

```
begin
    allocation sequences initialization (ASI);
    deadline division (DD);
    idle-time block searching (IBS);
    task and block mapping (TBM);
    schedule improvement (SI);
    return;
```

Each allocation sequence is a topological order of the workflow. Different topological orders are obtained based on different priorities of the tasks. Let the task allocation sequence be $\bar{s}$. Details of the SI are shown in Algorithm 2. The algorithm starts from the dummy start node v_0, the tasks with no predecessors are added to the eligible set ES. Each time the task with the highest priority is selected from ES and added to $\bar{s}$ and the complete set CS. Four different rules are used to calculate the priorities of tasks, which are listed as below.

Algorithm 2: Allocation sequences initialization (ASI)

```
begin
    CS ← ∅, ES ← ∅ s̄ ← ∅;
    repeat
        CS ← CS ⋃ {v_j};
        s̄ ← s̄ ⋃ {v_j} ;
        for each (v_j, v_k) ∈ E do
            if ∀(v_i, v_k) ∈ E, v_i ∈ CS then
                ES ← ES ⋃ {v_k};
        Using rules to select an task v_j from the Eligible set ES;
    until (v_n ∈ CS);
    return s̄;
```

- maximum upward rank value: The upward-rank value based priority considers both the structure of the workflow and the characteristics of the tasks. All the tasks are considered as the same and can be executed on each available instance. Details of the upward rank value calculation are shown in [32]. Tasks on the critical path will get a higher priority and will be processed first.

- minimal processing time: In this rule, only the characteristics of the task are considered. The tasks' priorities are determined by the processing time. The minimal processing time has the highest priority.

- maximum number of successors: In this rule, only the structures of the workflow are considered. The priorities of the tasks are determined by the number of successors.

- minimal slack time: In this rule, both the structure of the workflow and the characteristics of the tasks are considered. The slack time of a task v_i is denoted as $sl_i = lst_i - est_i$. The task with the smallest slack time has the highest priority.

- maximum upward rank value with preemptive tasks: Preemptive tasks are supposed to use the maximum amount of available resources and they would have the lowest upward rank value.

4.2. Deadline division

Deadline division is crucial for the deadline constrained workflow scheduling problem [16, 26, 33] in cloud computing. There are two main steps to divide the workflow deadline to the tasks: critical path searching and free time slots distributing. During the critical path searching procedure, the suitable virtual machine and the corresponding execution time for the task should be determined first. Based on the estimated execution time of each task, the critical path and partial critical path are calculated. For each partial critical path, there are some free time slots between the earliest finish time and the deadline of the workflow. These free time slots should be distributed to the tasks on the partial critical path. Thus, each task obtains a sub-deadline based on the free time slots distributing procedures.

4.2.1. *Critical path searching*

To search the critical path for the workflow, the estimated task execution time ET_i for each task v_i should be calculated first. The virtual machine instances with the minimum execution time are commonly adopted in existing works [26]. However, these virtual machine instances usually have a high unit cost, which leads to high total resource renting cost. Yuan et al. [15] (DET) and Abrishami et al. [16] (PCP) consider a balance between the performance and cost. Li can Cai [33] modeled the virtual machine instance selection problem with Integer Programming, and solve the problem with the IBM ILOGCPLEX solver. The virtual machine instances with proper performance and cost are selected. However, these algorithms are not suitable for the problem under study in this paper because spot block instances and preemptive tasks are considered.

In this paper, a new virtual machine selection strategy is developed for workflow applications with preemptive and non-preemptive tasks. Since only homogeneous virtual machine instances (the same configuration) are considered, the tasks can select any type of virtual machine instances. No matter the instance is an on-demand or spot block instance, the estimated task execution time is the same. For the preemptive task, it can be interrupted during execution, the task can be divided into several parts and select different virtual machine instances. Details of the VM selection are shown in Algorithm 3. The virtual machine selection is based on the topological order obtained in SI. For each task v_i in the sequence $\bar{s}$, the estimated execution time is calculated. If the task is v_i a non-preemptive task, the estimated task execution time ET_i equals to the the processing time of task v_i on a single virtual machine, i.e., $ET_i = P_i$. If the task is v_i a preemptive task, the task is divided averagely on all the available virtual machines, i.e., $ET_i = P_i/|ES|$.

After calculating the estimated execution time of each task, the critical and partial critical path is determined using the traversing method [16]. The earliest start time est_i, the earliest finish time eft_i, the latest start time lst_i and the latest finish time lst_i of each task v_i are calculated with the critical path based method. [34]. For each partial critical path, there are some free time slots between the estimated finish time and the

Algorithm 3: Virtual machine instances selection (VMS)

Input: the task allocation sequence $\bar{s}$.
Output: the estimated execution time of all the tasks in $\bar{s}$.
begin
  repeat
    $CS \leftarrow \emptyset$, $ES \leftarrow \emptyset$;
    $v_i \leftarrow$ the first task of $\bar{s}$;
    $CS \leftarrow CS \bigcup \{v_i\}$;
    for *each* $(v_i, v_k) \in E$ **do**
      if $\forall (v_j, v_k) \in E, v_j \in CS$ **then**
        $ES \leftarrow ES \bigcup \{v_k\}$;
    if $v_i \in \mathbb{M}$ **then**
      $ET_i = P_i / |ES|$;
    else
      $ET_i = P_i$;
    $\bar{s} = \bar{s} - \{v_i\}$;
  until *(*$\bar{s} = \emptyset$*)*;
  return;

latest finish time the tasks. These free time slots are distributed to each task suitably and the deadline of each task are calculated.

4.2.2. Free time slots distributing

Traditionally, three different methods are adopted to distribute the free time slots to tasks. Abrishami et al. [16] proposed a PCP algorithm for the free time slots distribution in grid computing. The time slots are allocated to the tasks in the partial critical path proportionally according to the estimated execution times. However, they only consider the free time slots between the estimated finish time of the last task and the latest finish time of the partial critical path. The free time slots between the tasks inside the partial critical path are not considered. In the next work of Abrishami et al.

[26], the algorithm $IC-PCPD2$ was proposed to allocate the free time slots. All the free time slots between the tasks in the partial critical path are considered. However, the free time slots distribution methods do not consider the relationship between different partial critical path. Li can Cai [33] proposed a float free time slot based distribution method MRH, which considered the influence of the distributed partial critical path. The free time slots are allocated to different partial critical path iteratively.

In this paper, we adapted MRH to DDP to make it available for both preemptive and non-preemptive tasks. If the task v_i is a non-preemptive task, the total float free time slots of a critical path T_{CP}^{float} and the allocated float free time slots $T_{v_i}^{dis}$ are calculated the same as MRH. If the task v_i is a preemptive task, the estimated execution time are dynamic changed with the current available number of virtual machine instances. So T_{CP}^{float} is dynamic changed with the changing of est_i and eft_i. There are also no necessary to consider allocating float free time slots to preemptive tasks when calculating $T_{v_i}^{dis}$ for they can easily be terminated and executed on other virtual machine instances.

4.3. Idle-time block searching (IBS)

After the deadline of each task are calculate in the procedure DD, the procedure idle-time block searching is adopted to calculated the available time slot for each task. The matrix $R = (r_{ij})_{D \times H^r}$ are used to denote the availability of resources, in which the row represents time slots and the column represents the virtual machines. i.e., $r_{ij} = 1$ means the virtual machine j at the time slot i is occupied. The idle time block is sub-matrix $R[i^{'}, \ldots, i^{''}; j^{'}, \ldots, j^{''}]$ of R, in which all the elements are 0.

In the procedure IBS, free VMs are searched first. For each task v_i, from the system current time CT_i to the deadline D_i of the task, different idle time blocks are calculated. Different types of instances have different pricing intervals. Let the current pricing interval of the on-demand or spot block instance r_j be CP_j. Busy VMs are searched next. If the task running on the VM is a preemptive task, the preemption is allowed to occur. It means that if the running task on the current VM is a preemptive task, it can be preempted and then the VM becomes free and can be used as the free VM. Otherwise, if the task running on the VM is a non-preemptive task, it cannot be preempted. The

estimated release time is calculated based on the estimated task runtime on this VM. If available idle blocks are obtained for the task v_i, they are added to the idle block list $idleList$. If no available idle block is found and v_i is a non-preemptive task, new virtual machine must be rented (the preemptive task can be divided and easily find available idle blocks). For the on-demand instance r_l, three methods are used to determine the renting period.

- minimal renting period, the renting period equals to the estimated task runtime ET_i, i.e., $CP_l = CT_i + ET_i$.
- maximal renting period, the renting period equals to the latest finish time lft_i.
- deadline based renting period, the renting period equals to the deadline.

For the spot block instance r_l, three options are used to determine the block time (range from 1 to 6).

- minimal block time, the block time equals to the estimated task runtime ET_i.
- maximal block time, the block time equals to 6.
- deadline based block time, the block time equals to the minimum number between the deadline and 6.

Details of the Idle-time block searching (IBS) are shown in Algorithm 4. Line 5-12 searches the idle time block for free VMs. Lines 13-25 searches idle time block for

315 busy VMs. Line 26-34 applies new VMs for non-preemptive tasks.

Algorithm 4: Idle-time block searching (IBS)

Input: v_i: task to be scheduled;

1 **begin**
2 $idleList \leftarrow \emptyset$;
3 $VM_{free} \leftarrow$ Set of free VMs among rented VMs;
4 $VM_{busy} \leftarrow$ Set of busy VMs among rented VMs;
5 **foreach** $r_j \in VM_{free}$ **do**
6 $ET_i \leftarrow$ Estimate task runtime of v_i;
7 $D_i \leftarrow$ deadline of v_i;
8 $CP_j \leftarrow$ current pricing interval of r_j;
9 $AT_i \leftarrow \min\{D_i, CP_j\}$;
10 **if** $CT_i + ET_i < AT_i$ **then**
11 Compute idle time IT_j for r_j;
12 Add r_j to $idleList$;
13 **foreach** $r_j \in VM_{busy}$ **do**
14 $ET_i \leftarrow$ Estimate task runtime of v_i;
15 $D_i \leftarrow$ deadline of v_i;
16 $CP_j \leftarrow$ current pricing interval of r_j;
17 $AT_i \leftarrow \min\{D_i, CP_j\}$;
18 $k \leftarrow$ running task on r_j;
19 **if** v_k *is a non-preemptive task* **then**
20 $RT_j \leftarrow$ estimated release time for r_j;
21 **else**
22 $RT_j \leftarrow 0$;
23 **if** $CT_i + ET_i + RT_j < AT_i$ **then**
24 Compute idle time IT_j for r_j;
25 Add r_j to $idleList$;
26 **if** $idleList = \emptyset$ **then**
27 **if** $v_i \in \mathbb{R}$ **then**
28 $ET_i \leftarrow$ Estimate task runtime of v_i;
29 **if** $ET_i > 6$ **then**
30 Applying a new on-demand virtual machine l;
31 Calculate idle time IT_l for r_l;
32 **else**
33 Applying a spot block virtual machine l;
34 Calculate idle time IT_l for r_l;
35 Add r_l to $idleList$;
36 **return** $idleList$;

4.4. Task and block mapping (TBM)

In the procedure task and block mapping (TBM), the tasks are mapped to the idle time block obtained in IBS. Algorithm 5 shows the steps of mapping a task to the suitable idle-time block. If the task is a preemptive task, the task can be terminated during its execution. So the task can be divided into several parts and mapping each part to a suitable idle-time block separately. Two rules are used to divide and map preemptive task.

- Longest mapping rule, sorting the length of all the available idle time block. The mapping rule starts from the longest idle time block. If the task's execution time is less than the block, the task is allocated to this block. If the task's execution time is more than the block, the task is divided into two parts, one part is allocated to the current block with the longest execution time and the other part is rescheduled to find the next longest idle time block.
- Average mapping rule, finding the entire available idle time block, dividing the task averagely on the entire idle time block.

If the task is a non-preemptive task, the task can only be allocated to exactly one idle-time block. Similarly, four rules are used for mapping a non-preemptive task to the suitable idle-time block.

- Shortest mapping rule, sorting the length of the entire available idle time block. v_i is mapped to the idle time block with the shortest length.
- Longest mapping rule, sorting the length of the entire available idle time block. v_i is mapped to the idle time block with the longest length.
- Random mapping rule, v_i is mapped to the idle time block randomly.
- Minimal cost rule, sorting the unit price of the idle time block, v_i is mapped to the idle time block with the minimal extra cost.

Algorithm 5: Task and block mapping (TBM)

Input: the task allocation sequence $\bar{s}$, the idle-time block list $idleList$.
Output: the current schedule π.
begin
  repeat
    $v_i \leftarrow$ the first task of $\bar{s}$;
    if $v_i \in \mathbb{M}$ **then**
      for *(*$j = 0; j < |idleList|; j \leftarrow j + 1$*)* **do**
        Divide task v_i into a series of subtasks. Mapping subtasks suitable idle-time blocks;
    if $v_i \in \mathbb{R}$ **then**
      for *(*$j = 0; j < |idleList|; j \leftarrow j + 1$*)* **do**
        Using rules to determine the suitable idle-time block for v_i;
    Update $idleList$;
    Remove the task of $\bar{s}$;
  until *(*$\bar{s} = \emptyset$*)*;
  return π;

4.5. Schedule improvement

After the schedule of the problem is constructed, the procedure schedule improvement SI is adopted to reduce the still available idle time blocks. SI reduces the available idle time blocks mainly by two procedures: backward and froward moving to concentrate the idle time blocks, reduce the amount of spot block instances for preemptive tasks. The two moving procedures are carried out cooperatively. First, all tasks are moving backward according to the non-increasing order of the finish times of the current schedule. The schedule is kept in the priority list $\mathcal{L}_B$. The head task $\mathcal{L}_B^{[1]}$ is removed from $\mathcal{L}_B$. All successors of $v_{[1]}$ have been calculated before the calculation of $v_{[1]}$ and precedence constraints are not checked. The start time t of $v_{[1]}$ is decreased one by one from $lst_{[1]}$ to $s_{[1]}$ If task $v_{[1]}$ is a preemptive task, the amount of on-demand resources is also decreased one by one. The corresponding resource renting costs are

calculated for all possible schedules. The start time and the resource amount with the minimum costs of $v_{[1]}$ are updated with the new ones. $v_{[1]}$ is removed from $\mathcal{L}_B$. The procedure is repeated until $\mathcal{L}_B$ is empty. Then all tasks are moving forward according to the non-decreasing order of start times of the current schedule. The decreasing strategy of start times is similar to the increasing strategy in the backward moving process. The new start time t is increased one by one from $s_{[1]}$ to $est_{[1]}$. Algorithm 6 shows the

details of the SI procure with backward moving.

Algorithm 6: Schedule improvement with backward moving (SI)

begin
  $\mathcal{L}_B \leftarrow$ Sort tasks in π by non-increasing order of finish times, $\pi^c \leftarrow \pi$;
  repeat
    $v_{[1]} \leftarrow \mathcal{L}_B^{[1]}$, $\pi' \leftarrow \pi$, $s'_{[1]} \leftarrow lst_{[1]}$, $t' \leftarrow s_{[1]}$;
    repeat
      if $v_{[1]} \in \mathbb{M}$ **then**
        $h \leftarrow$ on-demand resources of $v_{[1]}$;
        repeat
          Calculate est_n;
          if $est_n > D$ **then**
            break;
          $h \leftarrow h - 1$;
        until $h = 0$;
        Calculate $C(\pi')$;
        if $C(\pi') \leq C(\pi)$ **then**
          $C(\pi) \leftarrow C(\pi')$, $t' \leftarrow s'_{[1]}$;
        $s'_{[1]} \leftarrow s'_{[1]} - 1$;
    until $s'_{[1]} < s_{[1]}$;
    $s_{[1]} \leftarrow t'$;
    Remove $\mathcal{L}_B^{[1]}$ from $\mathcal{L}_B$;
  until $Lenth(\mathcal{L}_\mathcal{B}) = 0$;
  if $C(\pi^c) < C(\pi)$ **then**
    $C(\pi^c) \leftarrow C(\pi)$, $\pi^c \leftarrow \pi$;
  else
    return π^{best};

5. Experimental results

5.1. Parameter calibration

There are seven parameters or components in the proposed ITB methods which need calibration. In the deadline division (DD) procedure, four different strategies are used to divide the workflow deadline to tasks: PCP (0), IC-PCPD2 (1), MRH (2) and DDP (3). Five tasks priority calculation rules are described in the sequence initialization (SI) procedure: maximum upward-rank value (0), minimal processing time (1), maximum number of successors (2), minimal slack time (3) and maximum upward rank value with preemptive tasks (4). In the idle-time block searching (IBS) phrase, three strategies are used to determine the renting period of on-demand instance: minimal renting period (0), maximal renting period (1) and deadline based renting period (2). Three strategies are proposed to determine the block time of spot-block instance: minimal block time (0), maximal block time (1) and deadline based block time (2). The task and block mapping (TBM) include two components need to be calibrated: the mapping of preemptive tasks and the mapping of non-preemptive tasks. For the preemptive tasks, two alternatives are considered: longest mapping rule (0) and average mapping rule (1). For the non-preemptive tasks, four alternatives are considered: shortest mapping rule (0), longest mapping rule (1), random mapping rule (2) and minimal cost rule (3). Finally, in the procedure schedule improvement (SI), the ITB method can consider with SI (0) or without SI (1). In total, there are $4\times5\times3\times3\times2\times4\times2=2880$ different combinations of algorithms. All algorithms are coded in Java and run on a computer with an Intel i5-3470 CPU (4 cores, 3.1GHz) and 6GBytes of RAM memory.

5.1.1. Design of experiment

We use random workflow instances for the parameter calibration. The workflow generator Rangen [35] is used to generate different random workflow instances. Deadline of the workflow is supposed to be $D = Est_n \times \theta$, in which θ is a deadline factor and takes a value randomly from $\{1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0\}$ according to [36]. These values make the deadline from tight to loose. The number of tasks n is set as $\{10, 20, 50, 100, 200\}$. For each size n, 20 different workflow instances are

generated. The network complexity of the workflow is set at 1.8 according to [35]. The processing time of each task takes a value randomly from a uniform distribution $U(1, 100)$. Preemptive and non-preemptive tasks are assigned randomly.

Only homogeneous virtual machines (the same configuration) are considered, as computing hosts with different configurations (CPU cores, memory and bandwidth) can be virtualized to the same. To simulate the virtual machine resource in real clouds, The CloudSim toolkit [37] is used. The toolkit is extended to support on-demand and spot-block resources. We take the VM instances (m4.large Amazon EC2 [2]) as the simulation example, the price of the instance is set according to Amazon (Table 1). The processor speed of the cores of the VM instance is set to 2000 MIPS. Each VM requires 1GByte of RAM and 10 GBytes of storage while the bandwidth is set at 500 Mbps. The times required for starting a host and creating a VM are as they are negligible in comparison to the execution time of a task.

Table 1: Unit cost of spot and on-demand instances

Instance Type	vCPU	On-demand Hourly	Spot Block Hourly	
			1 hour	6 hours
m4.large	2	\$0.120	\$0.069	\$0.088

The Relative Percentage Deviation (RPD) is used to evaluate the effectiveness of the compared algorithms. For an instance i, let the final schedule obtained by the current algorithm be π_i and its corresponding cost $C_(\pi_i)$. If π_i^* is the best schedule for instance i among the compared algorithms, the RPD of the current algorithm for instance i is calculated as follows:

$$RPD_i = \frac{C(\pi_i) - C(\pi_i^*)}{C(\pi_i^*)} \times 100\% \tag{14}$$

We use the multi-factor analysis of variance (ANOVA) technique to calibrate the values of the different parameters. ANOVA takes RPD as the response variable. First, the three main hypotheses (normality, homoscedasticity, and independence of the residuals) are checked from the residuals of the ANOVA. All three hypotheses are acceptable within the usual margins. Since all the p-values in the experiments are very close

to zero, they are not reported in this paper due to space considerations. Instead, we directly report the means plots resulting from the multiple pairwise tests in order to check which levels or variants of the studied factors are statistically better than the others. Interactions between (or among) any two (or more than two) factors are not considered as the observed F-Ratios are small in comparison with single factors.

5.1.2. Parameter calibration results

Figure 2 shows the means plot with 95% Tukey HSD confidence intervals for the deadline division, the sequence initialization and the schedule improvement methods. The difference between different deadline division is statistically significant. The proposed preemptive task based deadline division (DDP) is the best one with the average RPD value about 8%. For the five sequence initialization methods, the difference is also statistically significant. We observe that the rule maximum upward rank value with preemptive tasks is the best one. It is a little better than the rule maximum upward-rank value and much better than the other three compared rules. For the schedule improvement methods, ITB with improvement method is much better than that of ITB without improvement. The average difference is a significant 30%. Therefore, in the following algorithm comparisons, we use the DDP strategy as the deadline division methods, the rule maximum upward rank value with preemptive tasks as the sequence initialization methods and improve the proposed ITB with schedule improvement methods.

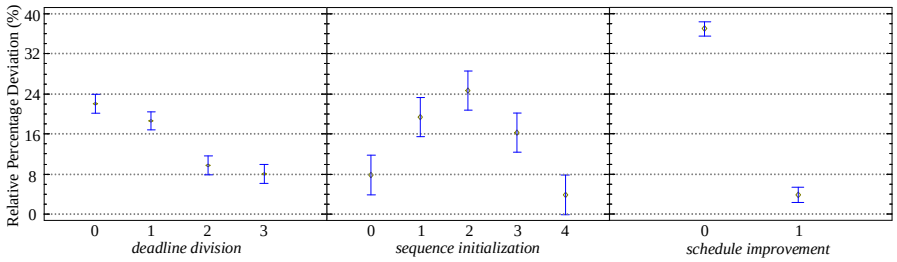

Figure 2: Means plot with 95% Tukey HSD confidence intervals for the deadline division, the sequence initialization and the schedule improvement methods.

The means plots with 95% Tukey HSD confidence intervals for the on-demand instance duration calculation, the spot instance block time determination, preemptive

tasks mapping rules and non-preemptive tasks mapping rules are shown in Figure 3. For the on-demand instance duration calculation, it can be observed that the differences are statistically significant. The methods with the minimal renting period is the best with the average RPD value 41%. It is better than the methods with the maximal renting period and deadline based renting period. For the spot instance block time determination, the difference is also statistically significant. The methods with the deadline based block time is the best with the average RPD value 43%. The methods with the maximal block time is the worst with the average RPD value 50%. For the two preemptive tasks mapping rules, the interval is overlapped. The difference between the two rules is not significant. The longest mapping rule is a little better than the average mapping rule. For the non-preemptive task mapping rules, the differences between the four rules are also not significant. The minimal cost rule is a little better than the other three rules with the average RPD value 42%. Therefore, we use the minimal renting period for the on-demand instance duration calculation, the deadline based block time for the spot instance block time determination, the longest mapping rule for the preemptive task mapping and the minimal cost rule for the non-preemptive task mapping in the following algorithm comparisons.

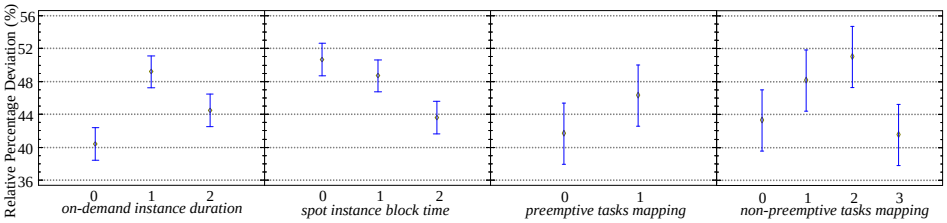

Figure 3: Means plot with 95% Tukey HSD confidence intervals for for the on-demand instance duration calculation, the spot instance block time determination, preemptive tasks mapping rules and non-preemptive tasks mapping rules.

5.2. Comparison with existing methods

5.2.1. Design of experiment

Since the problem of scheduling non-preemptive and preemptive tasks with on-demand and spot block instances in a workflow has not been studied yet, the just-in-

time algorithm (JIT) proposed by Poola et al. [29] is adapted for the considered problem. The ITB and JIT methods are also adapted to consider only on-demand resources. ITB_o is the proposed ITB with only on-demand resources. JIT_o is the just-in-time algorithm with only on-demand resources. In total, four algorithms ITB, ITB_o, JIT, JIT_o are compared. The ANOVA technique is also used to analyze the results in a sound and statistical way where RPD is the response variable.

The two scientific workflow instances Montage and LIGO [1] are adopted to analyze the effectiveness of the proposed MEFT in real environments. Montage has been created by NASA/IPAC [1] as an open source toolkit that can be used to stitch multiple input images together to create custom mosaics of the sky. The Laser Interferometer Gravitational Wave Observatory (LIGO) [1] is used to generate and analyze gravitational waveforms from data collected during the coalescing of compact binary star systems. Figure 4 and Figure 5 show an example of Montage and LIGO workflow applications. The nodes mProjectPP, mDiffFit and mJpEG in Figure 4 are regarded as the preemptive tasks while other nodes are non-preemptive tasks. The nodes TrigBank in Figure 5 are non-preemptive tasks and other nodes are preemptive tasks.

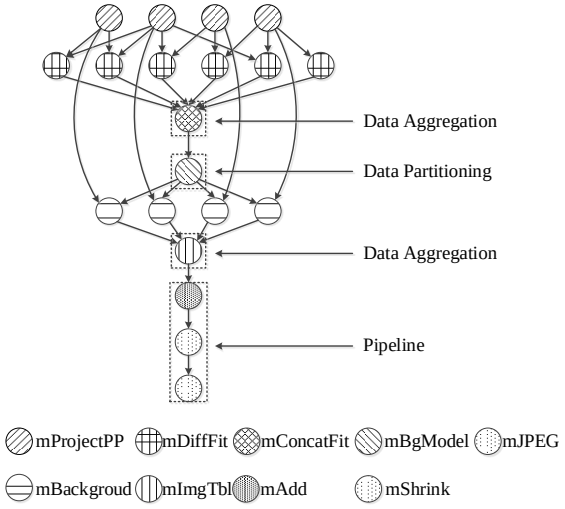

Figure 4: An example of Montage workflow application.

The instance size of each type of workflow application is set as $n \in \{50, 100, 200, 300, 400\}$.

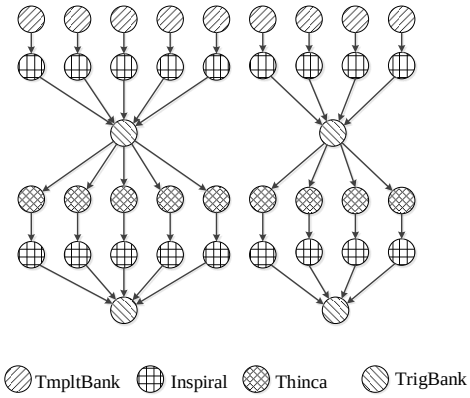

Figure 5: An example of LIGO workflow application.

The deadline factor takes a value from $\{1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0\}$. For each size and discount value 10 instances are generated. In total, there are $5 \times 10 \times 10 = 500$ instances for performance comparisons.

5.2.2. Comparison results of Montage instances

For Montage instances, the interaction plot between n and the compared algorithms with 95% Tukey HSD confidence intervals is shown in Figure 6. ITB is the best one among the compared algorithms. ITB outperforms JIT and ITB_o outperforms JIT_o for all the instances with different size. Comparing with JIT, ITB saves 10% cost on average. Comparing with ITB_o, ITB saves 20% cost on average than the ITB algorithm considering only on-demand instances. With the increase of n, the costs of ITB and JIT increase with the same gradient, e.g., $n = 400$, the RPD of ITB and JIT are about 65% and 55%, while those of ITB and JIT are about 25% and 32% when $n = 50$. This implies that the proposed ITB is much more suitable for the Montage applications with both smaller and bigger size.

The interaction plot between the deadline factors and the compared algorithms for Montage instances is shown in Figure 7. When the deadline factor is small, JIT is better than the proposed algorithm ITB, e.g., when $df = 1.1$, the RPD value of JIT is about 30%, while that for ITB is about 40%. With an increase in the deadline factor, the RPD value of JIT increases fast while that of ITB increases slowly. When $df > 1.2$, the

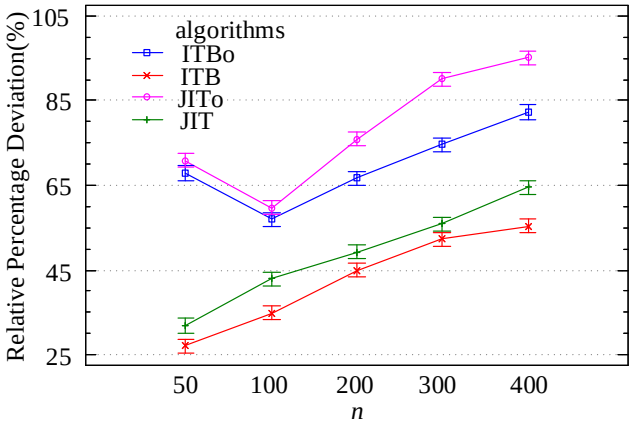

Figure 6: Comparison results of the algorithms on Montage applications with different n values.

proposed ITB performs better than JIT, e.g., when $df = 2.0$, the RPD value of JIT is about 72%, while that for ITB is about 48%. Comparing with ITB_o, ITB also saves about 20% cost on average. This implies that the deadline factor has little influence on the performance of the proposed ITB method but has a big influence on the compared JIT method.

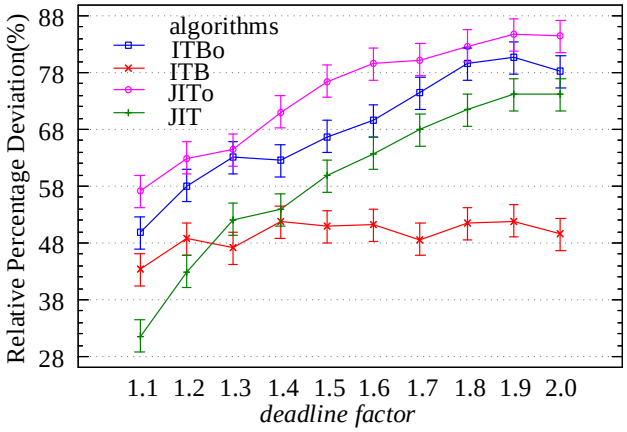

Figure 7: Comparison results of the algorithms on Montage applications with deadline factors.

5.2.3. *Comparison results of LIGO instances*

For LIGO instances, the comparison results are shown in Figure 8. We can observe that ITB also outperforms ITB_o, JIT and JIT_o for all the instances with different size. When n is small, the RPDs of ITB and JIT are overlapped. The performance differences between are not so significant, e.g., when $n = 50$, the RPD value of JIT about is 68%, while that for ITB is about 64%. With the increase of the instance size, the RPD decreases whereas that of JIT increases, e.g., when $n = 400$, ITB performs significantly better than JIT, the RPD values of JIT and ITB are about 70% and 60%. Comparing with ITB_o, ITB saves 20% cost on average. The RPD value of ITB_o also decreases while the instance size n increases. ITB is much more suitable for LIGO application with bigger size.

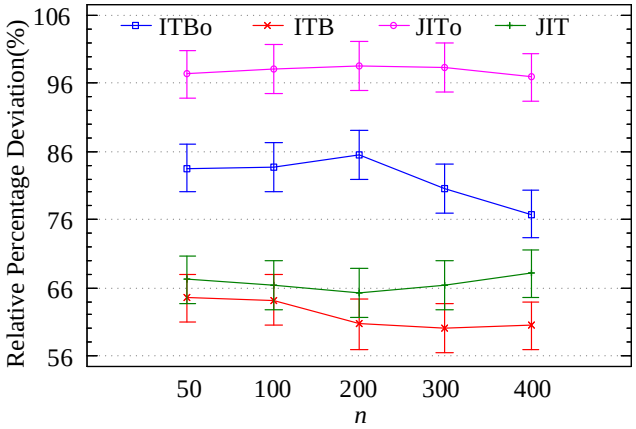

Figure 8: Comparison results of the algorithms on LIGO applications with different n values.

The comparison results with different deadline factors for LIGO instances are shown in Figure 9. We can observe that ITB performs better than other three algorithms for the entire deadline factor with different values. The proposed ITB saves about 10% cost than the JIT method and saves about 15% cost than than the ITB_o method. With an increase in the deadline factor, the RPDs of ITB, JIT and ITB_o keep almost the same. They have no monotone increase or decrease trend. However, the RPD of JIT_o increases while the deadline factor increases. The deadline factor also has little influence on the performance of the proposed ITB methods.

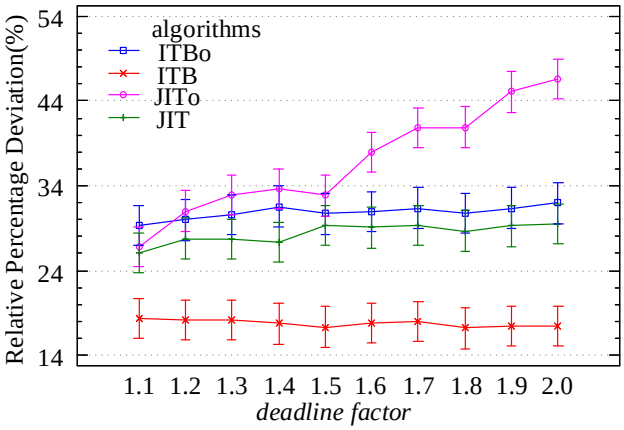

Figure 9: Comparison results of the algorithms on LIGO applications with different deadline factors.

The above comparisons of Montage and LIGO instances both show that the proposed ITB algorithm is much better than the JIT algorithm. The main reason lies in that the JIT algorithm uses task replication to enhance the reliability of workflow execution. The replication of tasks increase the total renting cost of resources. The ITB algorithm considered in this paper uses spot block instances to enhance the reliability of workflow execution. In fact, it will increase some resource renting cost. However, comparing to the replication of tasks, it saves a lot of cost.

6. Conclusion and Future Work

In this paper, a more realistic workflow scheduling problem with both spot block and on-demand instances was considered. A mathematic model with preemptive and non-preemptive tasks was established according to the two resource renting manners. We proposed a new idle time block based algorithm and compared it with the adapted JIT algorithm. The proposed ITB algorithm can easily be adapted to other workflow scheduling algorithms. The experiment results reveal that ITB got a better performance for all the instances with different size. For Montage instances, ITB is much more suitable for both bigger and smaller size and saves 25% cost on average for scheduling with only on-demand instances. For LIGO instances, ITB is much more suitable for bigger size and saves 30% cost on average for scheduling with only on-demand instances.

We would like to consider workflow scheduling with heterogeneous virtual machines and hybrid provisioning manners in future. Divide and allocate the preemptive tasks to heterogeneous virtual machines are much more complex than homogeneous virtual machines. The validations of the proposed algorithms in a real cloud environment (e.g., amazon EC2) are also promising topics.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (No. 61572127, 61272377), the National Key Research and Development Program of China (No. 2017YFB1400800). Rubén Ruiz is partially supported by the Spanish Ministry of Economy and Competitiveness, under the project "SCHEYARD – Optimization of Scheduling Problems in Container Yards" (No. DPI2015-65895-R) financed by FEDER funds.

References

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

[2] AmazonEC2, Amazon elastic compute cloud (Amazon EC2), http://aws.amazon.com/ec2/pricing.

[3] R. G. Michael, S. J. David, Computers and intractability: a guide to the theory of np-completeness, WH Free. Co., San Fr.

[4] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, et al., Pegasus: A framework for mapping complex scientific workflows onto distributed systems, Scientific Programming 13 (3) (2005) 219–237.

[5] M. Wieczorek, R. Prodan, T. Fahringer, Scheduling of scientific workflows in the ASKALON grid environment, ACM SIGMOD Record 34 (3) (2005) 56–62.

[6] Q. Chen, L. Wang, Z. Shang, Mrgis: A mapreduce-enabled high performance workflow system for gis, in: IEEE Fourth International Conference on eScience (eScience 2008), IEEE, 2008, pp. 646–651.

[7] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. P. Berman, P. Maechling, Scientific workflow applications on amazon EC2, in: 2009 5th IEEE International Conference on E-Science Workshops, IEEE, 2009, pp. 59–66.

[8] J. Yu, R. Buyya, Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms, Scientific Programming 14 (3) (2006) 217–230.

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

[10] A. Gerasoulis, T. Yang, On the granularity and clustering of directed acyclic task graphs, IEEE Transactions on Parallel and Distributed Systems 4 (6) (1993) 686–701.

[11] M. A. Palis, J.-C. Liou, D. S. L. Wei, Task clustering and scheduling for distributed memory parallel architectures, IEEE Transactions on Parallel and Distributed Systems 7 (1) (1996) 46–55.

[12] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, K. Kennedy, Task scheduling strategies for workflow-based applications in grids, in: IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005), Vol. 2, IEEE, 2005, pp. 759–767.

[13] W.-N. Chen, J. Zhang, An ant colony optimization approach to a grid workflow scheduling problem with various qos requirements, IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 39 (1) (2009) 29–43.

[14] J. Yu, R. Buyya, C. K. Tham, Cost-based scheduling of scientific workflow applications on utility grids, in: First International Conference on e-Science and Grid Computing (e-Science 2005), IEEE, 2005, p. 8.

[15] Y. Yuan, X. Li, Q. Wang, X. Zhu, Deadline division-based heuristic for cost optimization in workflow scheduling, Information Sciences 179 (15) (2009) 2562–2575.

[16] S. Abrishami, M. Naghibzadeh, D. Epema, Cost-driven scheduling of grid workflows using partial critical paths, IEEE Transactions on Parallel and Distributed Systems 23 (8) (2012) 1400–1414.

[17] Z. Cai, X. Li, J. N. D. Gupta, Critical path-based iterative heuristic for workflow scheduling in utility and cloud computing, in: Service-Oriented Computing, Springer, 2013, pp. 207–221.

[18] D. Klusácek, H. Rudová, Improving qos in computational grids through schedule-based approach, in: Scheduling and Planning Applications Workshop at the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008), Sydney, Australia, 2008.

[19] H. Liu, A. Abraham, V. Snášel, S. McLoone, Swarm scheduling approaches for work-flow applications with security constraints in distributed data-intensive computing environments, Information Sciences 192 (2012) 228–243.

[20] Y. C. Lee, R. Subrata, A. Y. Zomaya, On the performance of a dual-objective optimization model for workflow applications on grid platforms, IEEE Transactions on Parallel and Distributed Systems 20 (9) (2009) 1273–1284.

[21] H. M. Fard, R. Prodan, T. Fahringer, Multi-objective list scheduling of workflow applications in distributed computing infrastructures, Journal of Parallel and Distributed Computing 74 (3) (2014) 2152–2165.

[22] L. Chen, X. Li, R. Ruiz, Resource renting for periodical cloud workflow applications, IEEE Transactions on Services Computingdoi:10.1109/TSC.2017.2677450.

[23] L. Chen, X. Li, Cloud workflow scheduling with hybrid resource provisioning, The Journal of Supercomputing (2017, 10.1007/s11227-017-2043-5) 1–25doi:10.1007/s11227-017-2043-5.

[24] 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.

[25] 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.

[26] S. Abrishami, M. Naghibzadeh, D. H. Epema, Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds, Future Generation Computer Systems 29 (1) (2013) 158–169.

[27] 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. doi:10.1109/TSC.2014.2361320.

[28] D. Poola, K. Ramamohanarao, R. Buyya, Fault-tolerant workflow scheduling using spot instances on clouds, Procedia Computer Science 29 (2014) 523–533.

[29] D. Poola, K. Ramamohanarao, R. Buyya, Enhancing reliability of workflow execution using task replication and spot instances, ACM Transactions on Autonomous and Adaptive Systems (TAAS) 10 (4) (2016) 30.

[30] D. Jung, J. Lim, J. Gil, E. Lee, H. Yu, Task balanced workflow scheduling technique considering task processing rate in spot market, Journal of Applied Mathematics 2014.

[31] D. Jung, T. Suh, H.-C. Yu, J.-M. Gil, A workflow scheduling technique using genetic algorithm in spot instance-based cloud, TIIS 8 (9) (2014) 3126–3145.

[32] L. Chen, Y. Guo, X. Li, R. Ruiz, Hybrid resource provisioning for workflow scheduling in cloud computing, in: International Conference on Human Centered Computing, Springer, 2016, pp. 34–46.

[33] X. Li, Z. Cai, Elastic resource provisioning for cloud workflow applications, IEEE Transactions on Automation Science and Engineering 14 (2) (2017) 1195–1210.

[34] E. Demeulemeester, W. S. Herroelen, Project scheduling: a research handbook, Vol. 49, Kluwer Academic Pub, 2002.

[35] R. Kolisch, A. Sprecher, Psplib-a project scheduling problem library: Or software-orsep operations research software exchange program, European Journal of Operational Research 96 (1) (1997) 205–216.

[36] A. Drexl, A. Kimms, Optimization guided lower and upper bounds for the resource investment problem, Journal of the Operational Research Society (2001) 340–351.

[37] 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.