Resource renting for periodical cloud workflow applications

Long Chen, Xiaoping Li, *Senior Member, IEEE*, Rubén Ruiz

Abstract—Cloud computing is a new resource provisioning mechanism, which represents a convenient way for users to access different computing resources. Periodical workflow applications commonly exist in scientific and business analysis, among many other fields. One of the most challenging problems is to determine the right amount of resources for multiple periodical workflow applications. In this paper, the periodical workflow applications scheduling problem with total renting cost minimization is considered. The novelty of this work relies precisely on this objective function, which is more realistic in practice than the more commonly considered makespan minimization. An integer programming model is constructed for the problem under study. A Precedence Tree based Heuristic (PTH) is developed which considers three types of initial schedule construction methods. Based on the initial schedule, two improvement procedures are presented. The proposed methods are compared with existing algorithms for the related makespan based multiple workflow scheduling problem. Experimental and statistical results demonstrate the effectiveness and efficiency of the proposed algorithm.

Index Terms—Resource Allocation, Long-term Resources Renting, Periodical Multiple Workflows, Cloud Computing.

1 Introduction

Cloud computing is a novel market-oriented distributed computing model enabling convenient access to a pool of sharable computing resources (e.g., networks, servers, storage) potentially distributed and in a seamless and straightforward way. In the Cloud, resources are owned by Cloud Service Providers (CSP) and are encapsulated as services. Basically, users do not need to own resources which effectively spares them from expensive purchases and no less expensive run and maintenance costs. By renting resources from service providers for their requirements, users need to pay whenever they make use of the cloud computing services. In other words, cloud computing makes it convenient for users to access resources from anywhere and anytime with significant enhanced convenience and greatly reduced costs.

Generally, users' requirements are represented by workflow applications which describe a wide range of complex scientific and business analysis applications [1]. Besides complex precedences among tasks in workflow applications, there are many resource types, different resource provisioning alternatives and alternative renting resources which make workflow scheduling in cloud computing a hard to solve. In addition, most users lack professional skills and knowledge on how to rent resources from CSPs in a cost effective way (e.g., which resources are to be rented, how many and for how long). Therefore, cloud workflow scheduling is very complex. If this problem is not properly addressed, users end up spending large amounts of money to solve their cloud computing problems than needed.

Among workflow applications, periodical workflows are a typical type of applications which commonly exist in complex scientific and business analysis. For example, weather is forecasted every day; profit growth rates of companies are analyzed every month; gravitational waves in the universe are calculated every year, etc. (LIGO [2]). Such applications are periodically carried out, e.g., the weather forecasting model (workflow application) is calculated every hour to analyze the weather information and weather forecasting users collect the weather information every hour. Figure 1 shows the relationships between users and CSPs. Users send their periodical workflow applications to the CSPs. Each workflow in the period arrives at the system with some QoS (Quality of Service) constraints.

Usually workflow applications are constrained by deadlines, i.e., all tasks are scheduled to meet deadline requirements and they need to be appropriately assigned to CPSs' resources in order to minimize the total renting cost. Because of different characteristics, tasks require different types of resources (virtual machine instances). In addition, CSPs offer alternative services for each task. This means that each task might be performed in alternative ways or modes. Furthermore, tasks can be executed on several resources of the same or different types in parallel. Basically, depending of the type of resource we assign to a task, its execution time can be shortened or extended. Furthermore, tasks might be executed by different resources in parallel so in the end there are many potential combinations by which a task might be carried out. At the same time, resources can be shared between tasks inside a renting period, i.e., resources can be used by other tasks if the current task finishes during the resource renting period. Note that rented resources are paid by period and if a rented resource is not used for the whole period, that capacity is lost. Therefore, resource sharing that unused rented resources can be further used for other tasks. Therefore, tasks of complex workflow applications can be carried out in different ways or modes, using

Long Chen and Xiaoping Li are with the School of Computer Science and Engineering, Southeast University, Nanjing 211189, P.R. China, and also with the Key Laboratory of Computer Network and Information Integration, Southeast University, Ministry of Education, 211189, Nanjing, China (Tel: 86-25-52090916; Fax: 86-25- 52090916; e-mail: xpli@seu.edu.cn).
R. Ruiz is with Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Edifico 8G, Acc. B. Universitat Politècnica de València, Camino de Vera s/n, 46021, València, Spain (e-mail: rruiz@eio.upv.es).
Manuscript received ; revised .

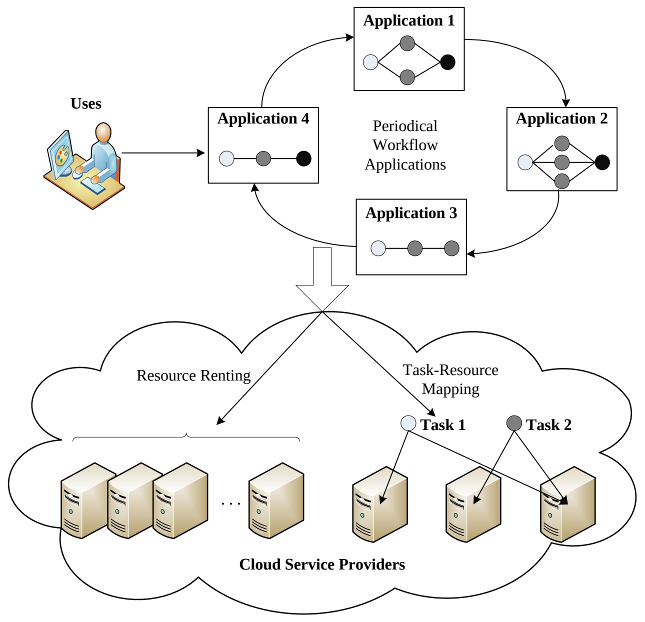


Fig. 1. Relationships between users and cloud service providers.

different types of cloud resources and potentially with different processing and completion times. As a result, there are a lot of potential savings for users if efficient methods are employed to appropriately select and rent the needed resources.

Generally, there are two resource renting alternatives provided by CSPs: on-demand and reserved [3]. The on-demand choice allows users to pay for computing capacity by the hour with short-term commitments which is suitable for real time applications. This liberates users from the costs and complexities of hardware planning, purchase and maintenance. Large fixed costs can be transformed into much smaller variable costs. The reserved option is a long-term strategy which enables users to make a low, one-time payment for each resource they need and keep the rented resources for a relatively long time. Users receive a significant discount on the hourly charge for each resource in the reserved option mainly due to the long time commitment. However, the resource utilization rate is usually low as it is hard to keep the rented resources busy all the time. This is a fact exploited by CSPs and is main reason why discounts are offered for long time reserved options. A different planning horizon results in different renting alternatives, i.e., long-term (year/months) planning with the reserved alternative and/or short-term (hour/minutes) with the on-demand alternative. Due to the nature of the considered periodical workflow applications, it is natural to adopt the reserved alternative as it is more cost effective to run in the long term. Renting the appropriate resources in different intervals of a time horizon is critical for minimizing the total resource renting cost of cloud computing resources.

In this paper, we consider scheduling periodical workflow applications on cloud resources with the objective of minimizing the total renting cost. This objective is closer to actual application scenarios, specially when compared to other more studied objectives that deal with the minimization of the completion time of the workflow application (makespan). This related makespan objective multiple workflow scheduling problem was studied in [4] and [5]. However, in the Cloud, while makespan can be reduced almost arbitrarily by renting more and more resources from the CSPs, users are usually concerned about total renting costs rather than in finishing their workflow applications as soon as possible. To the best of our knowledge, the problem under study is NP-hard and has not been considered yet in the literature. The main challenges are: (i) Determining the actual start times for multiple periodical workflow applications with different QoS constraints. (ii) Determining the appropriate amounts of reserved resources for tasks of different workflow applications. (iii) Mapping tasks with different modes to rented resources in order to minimize the total renting cost. The problem is mathematically modeled using Integer Programming. A Precedence Tree based Heuristic (PTH) is developed for the considered problem which consists of three components: workflow combination, initial schedule construction and schedule improvement. Three types of sink nodes are established for the workflow construction. Based on the enumeration tree scheme, a dynamic one-step rule considering both extra cost and freedom is developed for the schedule construction. Two main schedule improvement procedures considering both execution mode and resource type are presented. The multi-factor analysis of variance technique is used to find the best construction rules and improvement procedures. Comparisons with algorithms for the similar workflow scheduling problem show that PTH saves a lot of total renting cost for periodical workflow applications.

The rest of the paper is organized as follows. Related works are described in Section 2. Section 3 gives the definition and proposes a mathematical model for the LRRP. In Section 4, the PTH algorithm is described. Computational results are shown in Section 5, followed by conclusions and future research directions in Section 6.

2 Related Works

Workflow scheduling is a relevant problem both for users as well as for CSPs and therefore it has been studied thoroughly. Tasks are usually mapped to suitable resources so as to optimize some performance criterion. In the traditional distributed field (utility Grids), resources were usually geographically dispersed clusters. Resources were encapsulated as services and services were not shareable between tasks of the workflow. Only mapping was required and timetabling was not necessary during the scheduling process. There are two main objectives: cost optimization under deadline constraints and execution time optimization under budget constraints [6]. Common methods for time optimization include dynamic programming [7], branch and bound [8], decomposition-based methods [9], list scheduling algorithm [10], critical path based allocation [11], greedy randomized adaptive search [12] and ant colony optimization approach [13]. 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], market-oriented hierarchical scheduling strategy [19], workflow applications with security constraints [20], double auction-based scheduling of scientific applications [21] and workflow scheduling with multiple objectives [22][23]. However, only a few papers have focused on the scheduling of workflow applications in cloud computing, in which resources (virtual machines) were usually geographically concentrated and shared by tasks and workflows. In cloud computing, Alexandru Iosup et al. [24] analyzed the performance of cloud computing services for scientific computing workloads. Byun et al. [25] proposed a Balanced Time Scheduling (BTS) algorithm to allocate homogeneous resources to a workflow within a user-specified finish time according to the reserved strategy. Allocating heterogeneous resources to workflow applications was not considered. In a following work [26], the Partitioned Balanced Time Scheduling (PBTS) algorithm was presented for homogeneous resources in which resources were provided with the On-demand option. PBTS considered time partitions in the algorithm and minimized 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 the on-demand cases [27] in a cloud environment. The two proposed algorithms, IC-PCP and IC-PCPD2, were different from utility Grids in three ways: on-demand resource provisioning, homogeneous networks, and the pay-as-you-go pricing model. Followed by Cai et al. [28], the workflow scheduling problem with heterogeneous resources and On-demand resource provisioning was considered. The problem was divided 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 Match degree maximization) were developed for the sub-problems. Chaisiri et al. [29] formulated a stochastic programming model for the On-demand cloud resource provisioning with uncertain demand and price. Zuo et al. [30] proposed a self-adaptive learning particle swarm optimization for minimizing the cost of outsourcing deadline constrained tasks in hybrid IaaS clouds. The short-term resources provisioning for workflow applications with the on-demand option is usually considered in the existing literature.

In addition, little work has considered the multiple workflows scheduling problem. Xu et al. [4] proposed an algorithm for scheduling multiple workflows with several QoS constrains in cloud. Resources were encapsulated as services and are not sharable between workflows. Bittencourt et al. [5] proposed four different strategies for scheduling multiple workflows on fixed resources of Grids and evaluated them in terms of makespan and fairness. Each task of workflows was executed only on one resource (processor). Resources were limited and cost minimization was not considered. Therefore, the problem of scheduling multiple periodical workflow application with different arrival times has not been considered yet in cloud where resources are regarded as unlimited and shareable.

3 Problem Description and Formulation

A workflow application is commonly depicted by a task-on-node Directed Acyclic Graph (DAG), in which tasks are denoted by nodes and dependencies between tasks are represented by edges. Suppose there are g workflow applications $G = \{G^1, \dots, G^g\}$ arriving at the period D. Each workflow application G^w, $w \in \{1, \dots, g\}$ is represented by $G^w = (V^w, E^w)$. There are n^w tasks in the workflow application G^w, i.e., $V^w = \{v_1^w, \dots, v_{n^w}^w\}$. Edge $(v_i^w, v_j^w) \in E^w$ denotes the precedence relationship from task v_i^w to v_j^w. The arrival time and deadline of each workflow G^w are A^w and T^w respectively. Different types of resources (virtual machines) provided by the CSP are represented by $R = \{R_1, \dots, R_a\}$. The unit cost of resource R_k is c_k. For each task v_j^w of the workflow G^w, v_j^w can be executed by a mode M_{jo}^w. Each mode includes the corresponding processing time d_{jo}^w and the units of resources (r_{jok}^w units for R_k) required by v_j^w. Since resources are charged by hour, all the processing times are integer and measured in hours in this paper. We suppose that there are m^{jw} number of execution modes for the task v_j^w, i.e., v_j^w has $M_j^w = \{M_{j1}^w, \dots, M_{jm^{jw}}^w\}$ different execution modes.

Let the schedule π be a solution for the considered problem, in which H_k is the units of resource R_k allocated during the period. Since the reserved alternative is adopted, the total renting cost is calculated as $C(\pi) = \sum_{k=1}^{a} c_k H_k D$. Let x_{jot}^w be a binary variable. x_{jot}^w takes value one if the j-th task v_j^w, $j \in \{1, \dots, n^w\}$ of workflow G_w is carried out in the o-th mode M_{jo}^w, $o \in \{1, \dots, m^{jw}\}$ and starts at time t, $t \in \{0, \dots, T^w\}$. Then, the considered problem can be modeled as follows:

$$\min \sum_{k=1}^{a} c_k H_k D \tag{1}$$

s.t.

$$x_{jot}^w \in \{0,1\},\ \forall w \in \{1, \dots, g\},\ \forall j \in \{1, \dots, n^w\},\ \forall o \in \{1, \dots, m^{jw}\},\ \forall t \in \{0, \dots, T^w\} \tag{2}$$

$$H_k \geqslant 0,\ \forall k \in \{1, \dots, a\} \tag{3}$$

$$\sum_{o=1}^{m^{jw}} \sum_{t=0}^{T^w} x_{jot}^w = 1,\ \forall j \in \{1, \dots, n^w\},\ \forall w \in \{1, \dots, g\} \tag{4}$$

$$\sum_{o=1}^{m^{jw}} \sum_{t=0}^{T^w} x_{iot}^w (t + d_{io}^w) \leqslant \sum_{o=1}^{m^{jw}} \sum_{t=0}^{T^w} x_{jot}^w t,\ \forall j \in \{1, \dots, n^w\}\ \forall i \in \{1, \dots, n^w\}\ i \neq j,\ \forall w \in \{1, \dots, g\},\ \forall (v_i^w, v_j^w) \in E^w \tag{5}$$

$$\sum_{o=1}^{m^{jw}} \sum_{t=0}^{T^w} x_{1ot}^w t \geqslant A^w,\ \forall w \in \{1, \dots, g\} \tag{6}$$

$$\sum_{o=1}^{m^{jw}} \sum_{t=0}^{T^w} x_{n^w ot}^w (t + d_{n^w o}^w) \leqslant T^w,\ \forall w \in \{1, \dots, g\} \tag{7}$$

$$T^w \leqslant D,\ \forall w \in \{1, \dots, g\} \tag{8}$$

$$H_k \geqslant \sum_{w=1}^{g} \sum_{j=1}^{n_w} \sum_{o=1}^{m^{jw}} r_{jok}^w \sum_{\tau=t}^{t+d_{jo}^w-1} x_{jo\tau}^w,\ \forall k \in \{1, \dots, a\},\ \forall t \in \{0, \dots, D\} \tag{9}$$

Binary variables x_{jot}^w for each task are defined in Equation (2). Formula (3) ensures that the units of each resource R_k are nonnegative. Formula (4) ensures that each task of a workflow starts at exactly one time period and with only one mode. Precedence constraints for each workflow are given by Formula (5). The arrival time constraints of the workflow are presented in Formula (6). Each workflow can only start after the arrival time of the workflow. The deadline constraints of the workflow are denoted in Formula (7). Formula (8) shows the renting period constraint. Equation (9) specifies the resource availability constraints, in which $\sum_{\tau=t}^{t+d_{jo}^w-1} x_{jo\tau}^w$ is the task being processed at time t and H_k is the maximum amount of resource R_k over the whole renting period.

Figure 2 shows an example of the considered problem with two workflow applications and one resource type. The number to the right of the node represents the execution modes of each task. Each mode contains the required units of resource and the corresponding processing times (days). e.g., node v_1^2 of workflow G^2 can be executed with 3 units of resource for 6 days ($6 \times 24 = 144$ hours) or with 6 units of resource for 3 days. Each workflow arrives at time 0. The deadline of workflow G^1 is 30 days and that of G^2 is 25 days. Let the reserved period D be 30 days (1 month) and the unit cost of the resource be 5. Figure 3 shows an example schedule for this considered problem. After scheduling the tasks of workflow G^1, task 1 of workflow G^2 could start at (a), (b) or (c) with different modes Since resources are shared between workflows. Figure 4 shows the best solution for the example with 3 units of resource. The resource renting cost is $5 \times 3 \times 30 = 450$.

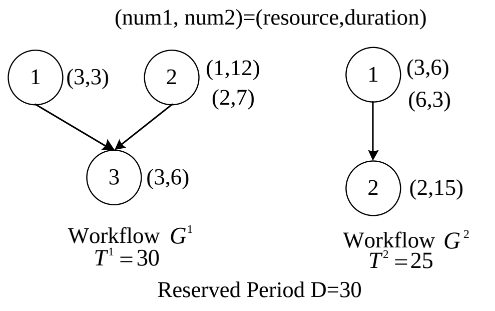


Fig. 2. An example of the considered problem.

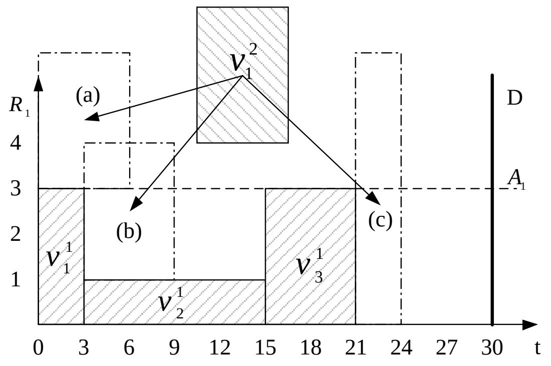


Fig. 3. A schedule for the example.

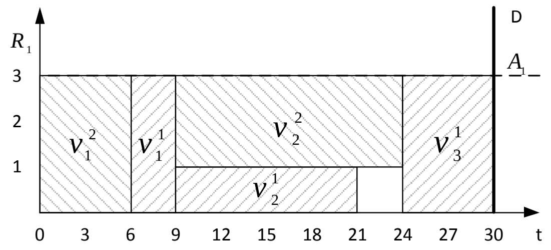


Fig. 4. The optimal schedule for the example.

In reality, the SaaS provider receives hundreds of workflows in a renting period. For each workflow, tens of tasks may be included. Hundreds of execution modes are available for each task. According to the mathematical model, the amount of binary variables x_{jot}^w is very large. Exact methods or meta-heuristics are impractical. Therefore, as the only realistic solution procedure, heuristic approaches are presented in the next section.

4 Precedence Tree based Heuristic

In this section, the Precedence Tree based Heuristic (PTH) is proposed for the considered problem. PTH consists of three phases: Workflows Combination and Parameters Initialization (WCPI), initial schedule Construction Methods (CM) and Schedule Improvement Procedure (SIP). WCPI considers the features and constraints of different workflows and combines them into a big single workflow. Relative parameters will be initialized and used in CM and SIP. Different types of rules are proposed in CM to construct the initial schedule for the considered problem. SIP contains two main improvement procedures, which decrease the resource renting cost by mode and resource adjustments.

The complete PTH include three main steps. For each step there are several different procedures. The full sketch of the algorithm is shown in Figure 5.

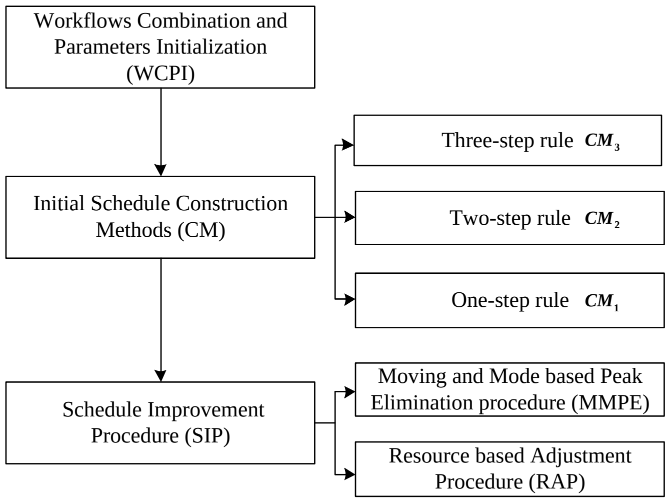

Workflows Combination and Parameters Initialization (WCPI)
Initial Schedule Construction Methods (CM)
Three-step rule CM_3
Two-step rule CM_2
One-step rule CM_1
Schedule Improvement Procedure (SIP)
Moving and Mode based Peak Elimination procedure (MMPE)
Resource based Adjustment Procedure (RAP)

Fig. 5. The sketch of the SCIP.

4.1 Workflows Combination and Parameters Initialization (WCPI)

Given a set of workflows with different resource requirements and deadline constraints, a Synchronization based Workflows Combination procedure (SWC) is proposed to combine multiple workflows into a big single workflow. Resources are unlimited and shareable between tasks of different workflows, tasks of different workflows can be considered as parallel tasks and be executed concurrently. In the synchronization procedure, two dummy synchronization nodes are added. The start synchronization node v_0^w of workflow G^w, $w \in \{1, \ldots, g\}$ is a node with zero resource requirements but with a processing time equal to A^w. The end synchronization node $v_{n^w+1}^w$ of workflow G^w, $w \in \{1, \ldots, g\}$ is a node with zero resource requirements but with a processing time equal to $D - T^w$. By adding the start and end synchronization node to the workflow, the start time of each workflow is synchronized to 0 while the deadline is synchronized to D. The characteristic (precedence and deadline constraints, resource requirements) of the workflow is not changed during the SWC. A dummy unique source node v_0 (with 0 resource requirements and 0 processing times) is added to all the workflows to enable them to start concurrently. Then, the dummy unique sink node v_{n+1} is appended to all workflows to make a big single workflow. Details of the procedure are formally described in Algorithm 1.

Using SWC, a big single workflow $G = (V, E)$ is obtained. The time complexity of SWC is $O(g)$, where g is the number of workflows.

Algorithm 1: Synchronization based Workflows Combination mechanism (SWC)

```
1  begin
2      $G \leftarrow (V, E)$, $V \leftarrow V^1 \cup V^w \cup \ldots \cup V^g$,
       $E \leftarrow E^1 \cup E^w \cup \ldots \cup E^g$;
3      Initialize $v_0$, $m^0 \leftarrow 1$, $d_{01} \leftarrow 0$,
       $r_{01k} \leftarrow 0, \forall k \in \{1, \ldots, a\}$;
       /* Initialization of the dummy source node $v_0$ */;
4      Initialize $v_{n+1}$, $m^{n+1} \leftarrow 1$, $d_{(n+1)1} \leftarrow 0$;
5      for $k = 1$ to $a$ do
6          $r_{(n+1)1k} \leftarrow 0$;
       /* Initialization of the dummy source node $v_{n+1}$ */;
7      $w \leftarrow 1$;
8      repeat
9          Initialize $v_0^w$, $m^0 \leftarrow 1$, $d_{01}^w \leftarrow A^w$;
10         for $k = 1$ to $a$ do
11             $r_{01k}^w \leftarrow 0$;
12         $E \leftarrow E \cup (v_0^w, v_1^w)$;
               /* Initialization of the start synchronization node */;
13         Initialize $v_{n^w+1}^w$, $m^{n^w+1} \leftarrow 1$,
           $d_{(n^w+1)1}^w \leftarrow D - T^w$;
14         for $k = 1$ to $a$ do
15             $r_{(n^w+1)1k}^w \leftarrow 0$;
16         $E \leftarrow E \cup (v_{n^w}^w, v_{n^w+1}^w) \cup (v_0, v_0^w) \cup (v_{n^w+1}^w, v_{n+1})$;
               /* Initialization of the end synchronization node */;
17         $w \leftarrow w + 1$;
18     until $(w > g)$;
19     return $G$;
```

Figure 6 shows an example of workflow combination. For the two workflows in Figure 2, three nodes are added, dummy source node, dummy sink node, and dummy synchronization node which are nodes 0, 7 and 6 respectively. The numbers at each node in the big workflow are rearranged for simplicity.

Let v_j be the tasks of G, the mode of v_j is simplified as $M_j = \{M_{j1}, \ldots, M_{jo^j}\}$. Resources requirements of M_{jo} are r_{jok} for each resource R_k, and the corresponding processing time is d_{jo}. Finding a schedule π for G consists of determining the start time $s_j = \sum_{t=0}^{D} x_{jot}t$ and execution mode $\mathcal{M}_j = \sum_{o=1}^{m^j} x_{jot}o$ of each task v_j. The finish time of task v_j is $f_j = s_j + d_{jo}$. $\mathcal{P}_j$ and $\mathcal{O}_j$ denote the immediate predecessor and immediate successor sets of v_j, i.e., $\mathcal{P}_j = \{v_i | \forall (v_i, v_j) \in E)\}$, $\mathcal{O}_j = \{v_k | \forall (v_j, v_k) \in E)\}$. Let est_j be the earliest start time of task v_j, lst_j be the latest start time of task v_j and $tlst_j$ be the latest threshold start time of task v_j. The earliest and latest start times are calculated based on the assumptions that all tasks are executed with the shortest duration mode (mode with minimal processing time). The earliest start time of task v_j is defined as $est_j = \max_{v_i \in \mathcal{P}_j}\{f_i\}$, and the latest start time of task v_j is defined as $lst_j = \min_{v_k \in \mathcal{O}_k}\{s_k - d_{jo}\}$. The latest threshold start time is calculated by using the longest duration mode for each task and is defined as $tlst_j = \min_{v_k \in \mathcal{O}_k}\{s_k - d_{jo}\}$. The interval $[est_j, lst_j]$ defines the time-window of the start time of task v_j.

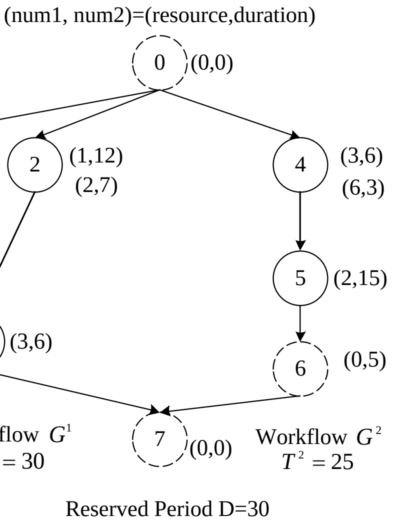


Fig. 6. An example of workflow combination.

v_j cannot start earlier than est_j and must start before lst_j in order to meet the deadline. $tlst_j$ represents the freedom or slack for the successors of v_j. If v_j starts before $tlst_j$, the successors of v_j are free to have other modes. Using the critical-path based Forward and Backward Pass Calculations [31], parameters est_j, lst_j, $tlst_j$ along with $\mathcal{P}_j$ and $\mathcal{O}_j$ of v_j $(\forall v_j \in V)$ can be obtained in $O(|E|)$ time. Relations betwen the temporal parameters of v_j are illustrated in Figure 7.

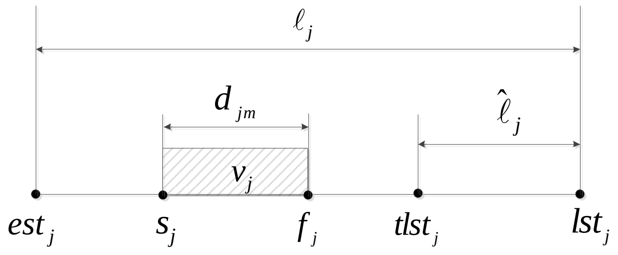


Fig. 7. The temporal parameters of v_j.

4.2 Initial Schedule Construction Methods (CM)

To construct a schedule π for the considered problem, the main aspect is to determine the execution mode and start time for each task. A precedence tree based enumeration scheme is established in this section to find one possible solution. Based on the enumeration scheme, different types of rules are proposed to construct a schedule.

4.2.1 Precedence Tree based Enumeration Scheme (PTES)

The precedence tree based enumeration procedure starts from the dummy start task v_0. Two sets including the complete set CS and the eligible set ES are calculated at each level of the enumeration tree. The complete set CS contains all scheduled tasks. Activities v_j with all the predecessors in the complete set CS are added to the eligible set ES. Then, a task v_j is selected from the eligible set ES and its relative mode M_{jo} and start time s_j are also determined. i.e., s_j takes a value from the interval $[est_j, lstj]$.

The solution for the considered problem is completed when the dummy end task v_{n+1} is added to the set ES. Details about the precedence tree based enumeration scheme are given in Algorithm 2.

Algorithm 2: Precedence Tree based Enumeration Scheme (PTES)

```
1  begin
2      $CS \leftarrow \varnothing, ES \leftarrow \varnothing$;
3      Compute $est_j$ and $lst_j$ for each task $v_j$;
4      Initialize state of the start task $v_j$,
         $j \leftarrow 0, \mathcal{M}_j \leftarrow 1, s_j \leftarrow 0$;
5      repeat
6          $CS \leftarrow CS \cup v_j$;
7          for each $v_k \in \mathcal{O}_j$ do
8              if $\mathcal{P}_k \subseteq CS$ then
9                  $ES \leftarrow ES \cup v_k$;
                 /* Update the complete set $CS$
              and eligible set $ES$ */;
10         Using heuristics rules: Select an task $v_j$ from the
             Eligible set $ES$, decide its execution mode $\mathcal{M}_j$
             and determine the appropriate start time
             $s_j \in [est_j, lstj]$;
11         for each $v_k \in \mathcal{O}_j$ do
12             $est_k \leftarrow \max\{est_k, s_j + d_{jo}\}$;
                /* For all the successors of $v_j$,
              dynamic update the earliest
              start time $est_k$ */;
13     until ($v_{n+1} \in CS$);
14     return;
```

To depict the precedence tree enumeration scheme for the considered problem, the simple example of Figure 6 is used. Figure 8 shows the procedure for node 1. Different selections of tasks from the eligible set result in different branches for the tree. The construction method starts from node 0 and explores a path from node 0 to node 7.

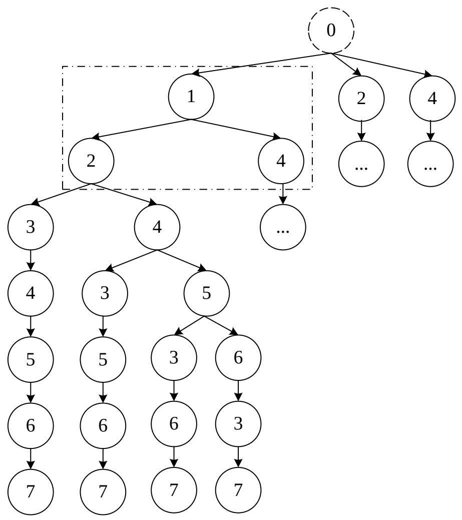


Fig. 8. Example of the Precedence Tree based Enumeration Scheme.

The enumeration scheme is an iterative process. At each iteration, the three branch factors (task, mode and start time) are determined. An inner branch is generated at each node of Figure 8. Take node 1 for example. The sketch is shown in Figure 9. Heuristic rules proposed in Line 7 of Algorithm 2 are used to determine the three branch factors which will lead to a better initial schedule.

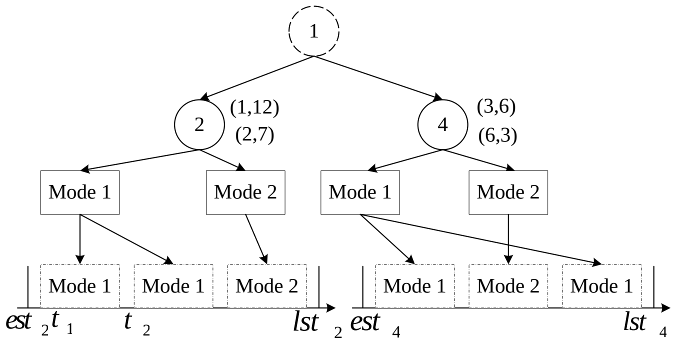


Fig. 9. Example of the inner branch.

4.2.2 Three-step Rule (CM_3)

The selection of the task, execution mode and start time results in a path from the root node to the leaf node in Figure 9. Generally, three-step rules are required to decide the three branch factors of the tree. Take Figure 9 for example, the three-step rule can first select an task v_2, and then mode 1 is selected from the possible execution modes of v_2. A time decision rule selects time t_2 as the start time. At every decision point, a branch is selected and other branches are not considered. So the three-step rule can find an acceptable solution quickly.

In fact, lots of rules can be used to select the three branch factors, and there are many possible combinations. In this paper, besides one three-step rule is established, we mainly focus on the one-step rules for effectiveness consideration. i) Activity decision: The minimal slack time rule ($\min_{v_j \in ES}\{\ell_j\}$) is used for task decision (Ties are broken arbitrarily). ii) Mode decision: The minimal product rule which selects the mode with the minimal product of processing time, resource demand and per unit cost is proposed (ties are broken arbitrarily). i.e., $\min_{o=1}^{m^j}\{\sum_{k=1}^{a} c_k d_{jo} r_{jok}\}$. iii) Start time decision: The Minimal Extra Cost (MEC) rule. Let the Partial Schedule PS be the schedule of the complete set CS during the enumeration process. Let h_{kt} be the requirements of resource R_k at time t in PS, and $H_k = \max_{t=0}^{D}\{h_{kt}\}$ be the maximum amount of resource R_k used in PS. The minimal extra cost rule is defined as $\min_{t=est_j}^{lst_j}\{\sum_{k=1}^{a} c_k \max\{H_k, r_{jok} + \max_{\tau=t}^{t+d_{jo}}\{h_{k\tau}\}\}\}$. (Ties are broken with the minimal start time).

Figure 9 shows an example of the minimal product rule for mode decision. After node 1, if node 2 is selected. Node 2 has two modes (1,12) and (2,7) in Figure 6. For mode 1, the product is $5 \times 1 \times 12 = 60$ (5 is the unit cost of the resource), and for Mode 2 it is $5 \times 2 \times 7 = 70$. So mode 1 of v_2 is selected. Figure 10 shows an example of the minimal extra cost rule for start time decision. The area in gray is the partial schedule PS. Node 4 can select a start time from est_4 to lst_4. MEC is calculated for each possible start time. When node 4 starts at time s_4, the minimal extra cost is obtained.

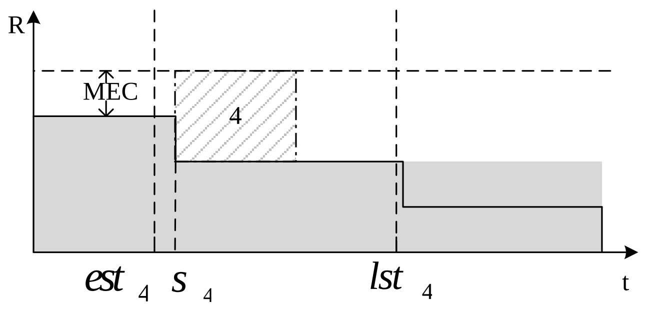


Fig. 10. Example of the minimal extra cost rule.

4.2.3 Two-step Rule (CM_2)

The three-step rule ignores the internal relationship between the three factors. Start time decision is commonly based on the execution mode. Therefore, some additional two-step rules are given in this section. The two-step rule first selects the task to be branched next, and then decides the mode and start time together.

As shown in Figure 7, two rules for task selection are established. Let the slack time ℓ_j of v_j be $\ell_j = lst_j - est_j$. The threshold slack time $\hat{\ell_j}$ is defined as $\hat{\ell_j} = lst_j - tlst_j$. The slack time ℓ_j defines the size of the available time window while the threshold slack time represents the freedom time window of v_j. The larger the $\hat{\ell_j}$, the bigger the effect on its successors, and the less freedom there is for the successors. The minimal slack time rule $\min_{v_j \in ES}\{\ell_j\}$ and the maximal threshold slack time rule $\max_{v_j \in ES}\{\hat{\ell_j}\}$ are adopted to select tasks. For mode and start time decisions, the Minimal Extra Cost rule (MEC) is also used. MEC in the two-step rule tests all available start times for all the possible modes and the best combination of mode and start time is selected. The two-step MEC is defined as $\min_{\forall m,t}\big\{\sum_{k=1}^{a} c_k \max\{H_k, r_{jok} + \max_{\tau=t}^{t+d_{jo}}\{h_{k\tau}\}\}\big\}$.

4.2.4 One-step Rule (CM_1)

The one-step rule sets task, execution mode and start time together based on visiting all leaf nodes in Figure 9. The maximal freedom rule and the minimal extra cost rule are modified to select all three factors at the same time. Each leaf node represents one selection of task v_j, mode M_{jo} and start time t.

Figure 7 demonstrates that if $t \leqslant dlst_j$, modes of successors of v_j are all available. If $t > tlst_j$, modes of successors of v_j depend on t and the execution mode M_{jo}. Therefore, the freedom rule is defined as $\min_{\forall j,m,t}\{t + d_{jo} - tlst_j\}$. The earlier the start time t and the smaller the processing time d_{jo}, the more freedom. Also, the minimal extra cost rule (MEC) is adapted to the one-step version. Unlike MEC in the three-step or two-step rule which just applies MEC to one or several leaf nodes in Figure 9, the MEC in the one-step rule explores all the possible leaf nodes and selects the best one. The one-step MEC is defined as $\min_{\forall j,m,t}\big\{\sum_{k=1}^{a} c_k \max\{H_k, r_{jok} + \max_{\tau=t}^{t+d_{jo}}\{h_{k\tau}\}\}\big\}$. For MEC, the less resource requirement and the longer execution time, the smaller extra cost.

There are some conflicts for the two rules in mode and start time decision. A single priority rule is proposed to integrate the advantage of the two one-step rules. First, the two rules are normalized so that they can be combined with each other. For the freedom rule, lst_j is the latest possible start time of t. Let m_f be the mode with the longest processing time of v_j. The freedom rule is normalized as $osr_1 = \min_{\forall j,m,t}\{(t + d_{jo} - dlst_j)/(lst_j + d_{jo_f} - dlst_j)\}$. Let the max value of MEC be $\text{MEC}_{\max}$. The minimal extra cost rule is normalized as $osr_2 = \min_{\forall j,m,t}\big\{\sum_{k=1}^{a} c_k \max\{0, r_{jok} + \max_{\tau=t}^{t+d_{jo}}\{h_{k\tau}\} - H_k)\}/(MEC_{\max} - \sum_{k=1}^{a} c_k H_k)\big\}$. The combined single priority rule is defined as $osr = (1-\beta)osr_1 + \beta osr_2$, in which β is the bias index of the rule. If β is close to 1, the more important the minimal extra cost rule is, and vice-versa. A dynamic β is also proposed based on the simple idea that the freedom rule often plays a big role at the beginning of the schedule construction while the minimal extra cost rule is preferred at the end. Let β be the ratio of the total cost of the Partial Schedule PS and the whole schedule. β is defined as $\beta = \sum_{v_j \in ES}\sum_{k=1}^{a} c_k r_{jok} / \sum_{v_j \in V} \max_{o=1}^{m^j}\{\sum_{k=1}^{a} c_k r_{jok}\}$. With the increase in the number of scheduled tasks in PS, β is increased from 0 to 1 during the schedule construction process.

4.3 Schedule Improvement Procedure (SIP)

Resources are always used in an unbalanced way in the initial schedule, i.e., a great number of resources are required in some periods while a small amount of resources are needed in other ones which may result in high resource renting costs. Therefore, the Schedule Improvement Procedure (SIP) is developed to balance the resource utilization. SIP consists of two processes: Moving and Mode based Peak Elimination procedure (MMPE) and Resource based Adjustment Procedure (RAP). The MMPE eliminates the peak demand of the resources by adjusting the schedule and mode of the initial solution. RAP reduces the renting cost by decreasing the amount of expensive resources and by increasing the amount of cheaper resources.

4.3.1 Moving and Mode based Peak Elimination procedure (MMPE)

The main idea of MMPE is to relocate tasks in the most resource demanding slots. The MMPE tries to relocate each task v_j between est_j and lst_j or to change its available execution mode in order to reduce the peak demand of the resource. In other words, tasks are moved from busy time slots to relatively idle time slots and eventually the total resource utilization is balanced.

There are two procedures inside MMPE: Backward Moving (BM) and Forward Moving (FM). In the Backward Moving, all tasks are sorted in a non-increasing order of the finish times of the current schedule and kept in priority list $\mathcal{L}_B$. The head task $\mathcal{L}_B^{[1]}$ (the current task with the biggest finish time) is denoted as $v_{[1]}$ and is removed from $\mathcal{L}_B$ (the second one becomes the head task $\mathcal{L}_B^{[1]}$ now). From the current point $s_{[1]}$ to $lst_{[1]}$, there are $lst_{[1]} - s_{[1]} + 1$ feasible starting points for $v_{[1]}$. The start time t is decreased one by one from $lst_{[1]}$ to $s_{[1]}$. Each available mode of $v_{[1]}$ is tried and relocated to the possible start point. Let the $H_k^{'}$ be the new the maximum amount of resource R_k. The corresponding resource renting costs $\sum_{k=1}^{a} c_k H_k^{'}$ are calculated for the possible schedules. The start time and the execution mode with the minimum costs are set as the new ones and the current $v_{[1]}$ is removed from $\mathcal{L}_B$. The procedure is repeated until $\mathcal{L}_B$ is empty. According to $\mathcal{L}_B$, all successors of $v_{[1]}$ have been calculated before the calculation of $v_{[1]}$. In other words, precedence constraints are always satisfied and there is no need to check them. Forward Moving performs in an opposite manner to BM. The priority list $\mathcal{L}_F$ is built according to the non-decreasing order of the start times of the current schedule obtained by BM. The decreasing strategy of start time is similar to the increasing one in BM, where the new start time t is increased one by one from $s_{[1]}$ to $est_{[1]}$. The start time and execution modes with the minimum costs are set as the new ones. MMPE starts from the initial schedule and iteratively conducts BM and FM until no better schedule can be found.

Let π^{best} be the best schedule found so far and π^c be the best solution of the current generation. For each schedule π, MMPE(π) is described in Algorithm 3 and 4.

Algorithm 3: Moving and Mode based Peak Eliminate procedure (MMPE-part1)

```
1  begin
2      $\mathcal{L}_B \leftarrow$ Sort tasks in $\pi$ by non-increasing order of finish times, $\pi^c \leftarrow \pi$;
3      repeat
4          $v_{[1]} \leftarrow \mathcal{L}_B^{[1]}$, Remove $\mathcal{L}_B^{[1]}$ from $\mathcal{L}_B$;
           /* Start from the first task of $\mathcal{L}_B$ */;
5          $\pi' \leftarrow \pi$, $s'_{[1]} \leftarrow lst_{[1]}$, $t' \leftarrow s_{[1]}$;
           /* $v_{[1]}$ is scheduled to its latest start time */;
6          repeat
7              $m'_{[1]} \leftarrow 1$;
               /* $v_{[1]}$ is executed with the first mode */;
8              repeat
9                  Calculate $C(\pi')$;
10                 if $C(\pi') \leq C(\pi)$ then
11                     $C(\pi) \leftarrow C(\pi')$, $m_{[1]} \leftarrow m'_{[1]}$, $t' \leftarrow s'_{[1]}$;
12                 $m'_{[1]} \leftarrow m'_{[1]} + 1$;
                   /* Update the mode of $v_{[1]}$ */;
13             until ($m'_{[1]} > m^{[1]}$);
14             $s'_{[1]} \leftarrow s'_{[1]} - 1$;
               /* Moving $v_{[1]}$ forward, update the start time of $v_{[1]}$ */;
15         until ($s'_{[1]} < s_{[1]}$);
16         $s_{[1]} \leftarrow t'$;
17     until ($Lenth(\mathcal{L_B}) = 0$);
18     if $C(\pi^c) < C(\pi)$ then
19         $C(\pi^c) \leftarrow C(\pi)$, $\pi^c \leftarrow \pi$;
20     else
21         Go to step part2-2;
```

Algorithm 4: Moving and Mode based Peak Eliminate procedure (MMPE-part2)

```
1  begin
2      $\mathcal{L}_F \leftarrow$ Sort tasks in $\pi$ by non-decreasing order of start times;
3      repeat
4          $v_{[1]} \leftarrow \mathcal{L}_F^{[1]}$, Remove $\mathcal{L}_F^{[1]}$ from $\mathcal{L}_F$;
           /* Start from the first task of $\mathcal{L}_F$ */;
5          $\pi' \leftarrow \pi$, $s'_{[1]} \leftarrow est_{[1]}$, $t' \leftarrow s_{[1]}$;
           /* $v_{[1]}$ is scheduled to its earliest start time */;
6          repeat
7              $m'_{[1]} \leftarrow 1$;
               /* $v_{[1]}$ is executed with the first mode */;
8              repeat
9                  Calculate $C(\pi')$;
10                 if $C(\pi') \leq C(\pi)$ then
11                     $C(\pi) \leftarrow C(\pi')$, $m_{[1]} \leftarrow m'_{[1]}$, $t' \leftarrow s'_{[1]}$;
12                 $m'_{[1]} \leftarrow m'_{[1]} + 1$;
13             until ($m'_{[1]} > m^{[1]}$);
14             $s'_{[1]} \leftarrow s'_{[1]} + 1$;
               /* Moving $v_{[1]}$ backward, update the start time of $v_{[1]}$ */;
15         until ($s'_{[1]} > s_{[1]}$);
16         $s_{[1]} \leftarrow t'$;
17     until ($Lenth(\mathcal{L_F}) = 0$);
18     if $C(\pi^c) < C(\pi)$ then
19         $C(\pi^c) \leftarrow C(\pi)$, $\pi^c \leftarrow \pi$, Go to Step part1-2;
20     if $C(\pi^c) < C(\pi^{best})$ then
21         $\pi^{best} \leftarrow \pi^c$, $C(\pi^{best}) \leftarrow C(\pi^c)$;
22     return $\pi^{best}$;
```

4.3.2 *Resource based Adjustment Procedure (RAP)*

RAP contains two main stages which adjust the resource amounts between different types of resources with different prices. One stage decreases the amount of expensive resources and the other increases the use of cheap resources.

In the decreasing stage, resources are sorted in descending order of their unit costs. For a solution π, let H_k be the resource usage amount of each resource R_k. The amount of resource R_k is reduced one by one. The solution is rescheduled based on the current resource available amount H_k. In this case, the schedule π might be unfeasible for some workflows exceeding their deadline. The Feasibility Verification Procedure (FVP) is proposed to verify the feasibility of the current H_k. FVP also adopts the Precedence Tree based Enumeration Scheme (PTES). The two step rule is used. The minimal slack time rule is also used to select tasks while the earliest finish time is adopted to select mode and start times. If the finish time of sink end node f_{n+1} is smaller than the reserved period D, the current H_k is feasible. The procedure is repeated until an unfeasible schedule is obtained. The increasing stage is then performed. During this stage, all the tasks v_i are sorted in decreasing order of $lst_i - s_i$. Activity v_j with the maximal $lstj - s_j$ indicates that s_j is not particularly relative to lst_j and v_j is selected as the increasing task. Suppose the execution mode of v_j is $\mathcal{M}_j$, the current H_k is increased to $H_k + r_{j\mathcal{M}_jk}$ for each resource R_k. Then FVP is called again to verify the feasibility. If H_k is not feasible, the task with the second biggest $lstj - s_j$ is selected as the increasing task. If H_k is feasible, the decreasing stage is invoked. The procedure stops when no better solutions can be found. Details of the resource based adjustment procedure are given in Algorithm 5.

5 Computational Results

To the best of our knowledge, there's no workflow scheduling problem proposed in the literature with the same features as the

Algorithm 5: Resource based Adjustment Procedure (RAP)

```
1  begin
2      Initialize $\pi$, $H_k$;
3      if $FVP(H_k)$ is false then
4          Go to step 13;
           /* Check if $H_k$ is feasible */;
5      Compute $lst_i - s_i$ for each task $v_i$. Select task $v_j$ with
         the maximum $lst_j - s_j$;
         /* If not feasible, increase $H_k$ two
       times */;
6      $H_k \leftarrow H_k + r_{j\mathcal{M}_j k}, \forall k \in \{1, \ldots, a\}$;
7      if $FVP(H_k)$ is true then
8          Go to step 12;
9      Select task $v_j$ with the second maximum $lst_j - s_j$;
10     $H_k \leftarrow H_k + r_{j\mathcal{M}_j k}, \forall k \in \{1, \ldots, a\}$;
11     if $FVP(H_k)$ is false then
12         return;
13     $k \leftarrow 1$;
14     repeat
15         $H_k \leftarrow H_k - 1$;
             /* If feasible, decrease $H_k$ */;
16         if $FVP(H_k)$ is false then
17             $k \leftarrow k + 1$;
18             Go to step 5;
19     until $k > a$;
20     return;
```

considered problem. To test the performance of the proposed algorithm, methods for the related makespan based multiple workflows scheduling problem [5] are adapted for the problem considered in this paper. All the algorithms are implemented in Java and executed on the same virtual machine with Intel i5-3470 CPU (4 cores, 3.1GHz) and 1GB memory of RAM.

Being a new problem, there are no existing benchmarks for the considered problem. To fairly compare different procedures, instances are randomly generated according to the characteristic of the problem. For the workflow, two important factors are considered: the number of tasks n and the network complexity NC. $n \in \{10, 20, 30, 40\}$ is the number of tasks in each workflow. NC is the average number of immediate successors of an task and is randomly generated from a uniform distribution $U[1, 3]$. For the resource, three important factors are considered: the types of resources $a \in \{2, 4, 6, 8\}$, the unit cost c of each resource type and the resource factor RF. c is randomly drawn from $U[1, 10]$. RF is the average number of types of resources needed by a task. We assume that RF takes a value randomly from $U[1, a]$. For the execution mode, there are three factors: the number of modes $o \in \{5, 15, 25, 35\}$, the resource requirement r and the processing time d. r and d are both randomly generated from $U[1, 10]$ (measured in hours). For the multiple workflows, four factors are considered: the renting period $D = 30 \times 24 = 720\ hours$, the number of workflows $g \in \{50, 100, 150, 200\}$, the start time A^w and the deadline T^w of each workflow G^w. Let $I^w = tlst^w_{n^w+1}$ (each task of G^w with the longest duration mode) be the maximal processing time of a workflow G^w. The start time A^w is randomly generated from $U[0, D - I^w]$. Accordingly, T^w is set as $A^w + I^w$. Generally, the number of tasks n, resource types a, modes o and workflows g are the main factors. For each combination of the above 4 factors, 10 instances are generated. Therefore, $4 \times 4 \times 4 \times 4 \times 10 = 2560$ instances are created in total.

There are three main different comparisons in this section. First, the different initial schedule construction methods are compared. Based on the construction methods, different schedule improvement procedures are tested next. The best resulting combination procedure (PTH) is compared with the existing method. As the tested algorithms are for long-term resource renting, only the performance (cost) is taken into account. The CPU time of each algorithm is not considered. RPD (Relative Percentage Deviation) is adopted to evaluate the performance. Let $f(i)$ be a cost obtained by algorithm f of instance i and $f^*(i)$ be the best known result for instance i. RPD is defined as:

$$RPD = \frac{f(i) - f^*(i)}{f^*(i)} \times 100 \qquad (10)$$

5.1 Comparisons of the Initial Schedule Construction Methods

Different initial schedule construction methods are compared in this section. There are ten different construction methods in total: one three-step heuristic CM_3; two two-step heuristics CM_2^1 and CM_2^2; six one-step heuristic CM_1^β ($\beta \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}$); and a dynamic one-step heuristic CM_1^d.

Table 1 shows the results of the ten initial schedule construction methods. From the average RPD, it can be easily seen that CM_1^d is the best algorithm among all the comparing algorithms for each size of instance. The one-step construction method is better than the two-step construction methods and much better than the three-step construction methods. For the two-step construction methods, CM_2^1 (minimal slack time, with the average 47.4) is better than CM_2^2 (maximal threshold slack time, with the average 65.3). CM_1^d is the best one-step rule and $CM_1^{0.6}$ is the second best one-step rule (with the average 5.4). While $\beta = 0$ represents rule osr_1 and $\beta = 1$ represents rule osr_2, the fact that CM_1^1 (with the average 17.1) is better than CM_1^0 (with the average 24.5) indicates that the minimal extra cost rule is better than the maximal freedom rule.

TABLE 1
Comparison of the Initial Schedule Construction Methods. RPD measure employed.

g	n	CM_1^0	$CM_1^{0.2}$	$CM_1^{0.4}$	$CM_1^{0.6}$	$CM_1^{0.8}$	CM_1^1	CM_1^d	CM_3	CM_2^1	CM_2^2
50	10	21.8	12.1	6.7	4.9	9.0	16.5	**0.0**	134.1	54.1	39.0
	20	24.1	13.2	6.1	5.1	8.6	17.6	**0.0**	138.6	56.9	41.5
	30	23.3	13.4	7.0	5.5	10.0	17.0	**0.0**	147.0	60.4	40.6
	40	22.3	13.2	6.7	5.4	9.1	17.5	**0.0**	153.3	61.2	44.3
100	10	23.4	11.4	7.2	5.1	9.6	17.0	**0.0**	140.2	61.6	41.2
	20	25.7	12.5	6.6	5.2	9.1	18.1	**0.0**	144.8	64.6	43.7
	30	24.9	12.6	7.5	5.7	10.6	17.5	**0.0**	153.4	68.3	42.8
	40	23.9	12.4	7.2	5.6	9.7	18.0	**0.0**	159.9	69.1	46.6
150	10	23.7	11.7	7.1	5.2	9.4	16.0	**0.0**	152.3	63.5	47.5
	20	26.1	12.8	6.5	5.4	8.9	17.1	**0.0**	157.1	66.5	50.1
	30	25.2	12.9	7.4	5.8	10.4	16.6	**0.0**	166.2	70.2	49.2
	40	24.3	12.7	7.1	5.7	9.5	17.1	**0.0**	173.0	71.1	53.1
200	10	24.8	11.9	7.2	5.2	10.0	16.5	**0.0**	171.0	65.0	52.2
	20	27.2	13.0	6.7	5.3	9.5	17.6	**0.0**	176.1	68.0	54.8
	30	26.3	13.1	7.6	5.8	11.0	17.0	**0.0**	185.8	71.8	53.9
	40	25.3	12.9	7.2	5.7	10.1	17.5	**0.0**	193.2	72.6	57.9
Average		24.5	12.6	7.0	5.4	9.7	17.2	**0.0**	159.1	65.3	47.4

To analyze the performance of each construction method on different instance factors in detail, a multi-factor analysis of variance (ANOVA) method is carried out. The response variable is the RPD. First, the three main hypotheses (normality, homoscedasticity, and independence of the residuals) are checked from the residuals of the experiments. All three hypotheses are acceptable from this analysis. Since all the p-values in the experiments are close to zero, they are not given in this paper. Greater F-Ratios imply factors with stronger effects. Interactions between (or among) any two (or more than two) factors are not considered because the observed F-Ratios are small in comparison.

The Means plot and Tukey HSD intervals of the one-step heuristic CM_1^{β} at the 95% confidence level are shown in Figure 11. $CM_1^{0.6}$ and CM_1^{d} are the best and are selected for the comparison with the other construction methods. The Means plot and Tukey HSD intervals at the 95% confidence level for instances with different workflow and task numbers are shown in Figure 12. For all the instances with different workflow and task values, we observe similar trends. CM_1^{d} is the best while CM_3 is the worst. The performance of the two one-step heuristics does not significantly change while the performance of the two-step or three-step heuristics become worse with an increase in the number of workflows or tasks. Figure 13 shows the Means plot and Tukey HSD intervals at the 95% confidence level for instances with different values of resource types and modes. CM_1^{d} is still the best and CM_3 is the worst. With the exception of the three-step heuristic which becomes worse as the number of modes increases, all the construction methods are not sensitive to the increase in the number of resource types or modes.

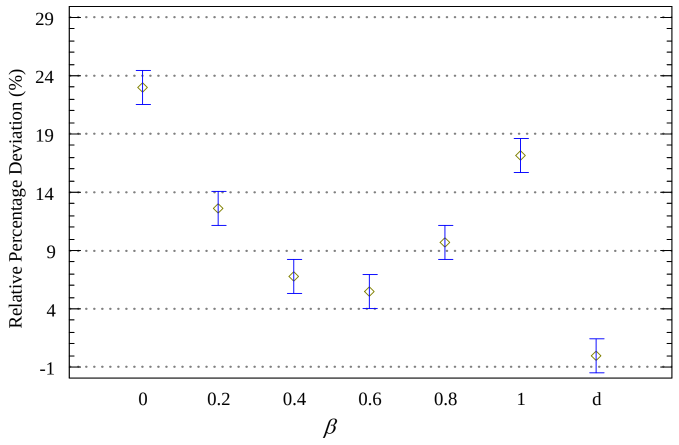


Fig. 11. Means plot and Tukey HSD intervals at the 95% confidence level for CM_1^{β}.

5.2 Comparisons of the Schedule Improvement Procedures

Comparisons of the schedule improvement procedure are also based on the generated instances. The best three methods for each category CM_3, CM_2^2 and CM_1^d are adopted as the initial schedule construction methods. Three schedule improvement procedures MMPE, RAP and the combination of MMPE with RAP are used to improve the initial schedules. SIP_i^j is used to denote each Schedule Improvement Procedure (SIP), in which i is the step of Construction Methods (CM) and j is the improvement procedure ($j = 1$ for MMPE, $j = 2$ for RAP and $j = 3$ for MMPE and RAP). Including the best initial schedule construction methods CM_1^d, there are 10 methods for comparing in total.

The Means plot and Tukey HSD intervals at the 95% confidence level of the ten compared methods are shown in Figure 14. $SIP_i^3, i \in \{1,2,3\}$ are much better than the other methods, which implies that SIP has a greater influence on the performance of the schedule construction and improvement procedure. The Means plot and Tukey HSD intervals at the 95% confidence level for instances with different workflow and task values are shown in Figure 15. There are similar trends for the instances with different workflow and task values. SIP_1^3 is the best while SIP_3^2 is the worst. The performance of each algorithm becomes worse with an increase in the number of workflows or tasks. Figure 16 shows the Means plot and the Tukey HSD intervals at the 95% confidence level for instances with different numbers of resource types and modes. The MMPE $SIP_i^2, i \in \{1,2,3\}$ are much more sensitive to an increase in the number of execution modes while RAP $SIP_i^3, i \in \{1,2,3\}$ are much more sensitive to an increase in the types of resource. Although SIP_1^3 is still the best and SIP_3^2 is the worst, this is not the case for other procedures with a different number of resources types and modes. e.g., SIP_1^2 is better than SIP_3^3 when $a = 8$ while it is worse than SIP_3^3 when $o = 35$.

5.3 Comparisons with existing methods

The best proposed algorithm SIP_1^3 is compared with the adapted algorithms. The CloudSim toolkit developed by Calheiros et al. [32] is used to simulate resource provisioning and scheduling in a real public cloud, which is also extended to support periodical workflow applications. Parameters used in CloudSim are listed as follows: The processor speed of each host is considered to be 2000 MIPS. Nine general purpose types of VM instances from Amazon EC2 are modeled (m4.large, m4.xlarge, m4.2xlarge, m4.4xlarge, m4.10xlarge, m3.medium, m3.large, m3.xlarge, m3.2xlarge). The CPU cores of each VM are listed in Table 2. Each VM requires 1024 MB of RAM and 10 GB of storage while the bandwidth is set as 500 B/S. The times required for starting a host and creating a VM are ignored for they are negligible comparing to the execution time of a task. The price of on-demand and reserved virtual machines resemble those of Amazon EC2. The virtual machine instances are assumed to be reserved with no upfront cost. The unit cost of reserved and on-demand instances are shown in Table 2.

TABLE 2
Unit cost of reserved and on-demand instances

Instance Type	vCPU	Cost Per Hour		Discount
		On-demand	Reserved	
m4.large	2	\$0.120	\$0.083	31%
m4.xlarge	4	\$0.239	\$0.164	31%
m4.2xlarge	8	\$0.479	\$0.329	31%
m4.4xlarge	16	\$0.958	\$0.658	31%
m4.10xlarge	40	\$2.394	\$1.645	31%
m3.medium	1	\$0.067	\$0.048	28%
m3.large	2	\$0.133	\$0.095	29%
m3.xlarge	4	\$0.266	\$0.190	29%
m3.2xlarge	8	\$0.532	\$0.380	29%

Workflow Applications: The two scientific workflow applications, Montage and LIGO [33] are adopted to analyze the effectiveness of the proposed PTH in real environments. Figures 17 and 18 show an example of Montage and LIGO workflow

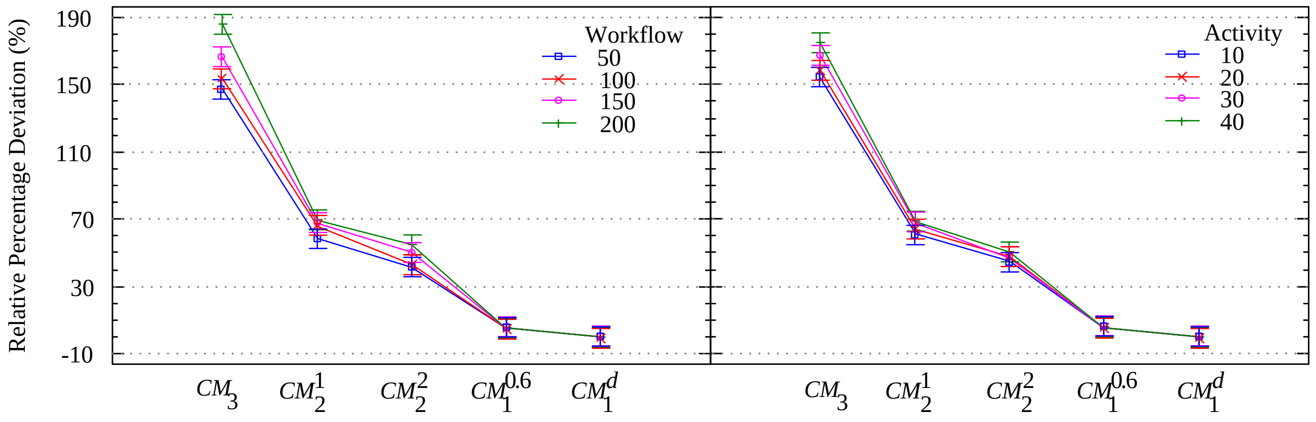


Fig. 12. Means plot and Tukey HSD intervals at the 95% confidence level for CM for instances with different number of workflows and tasks.

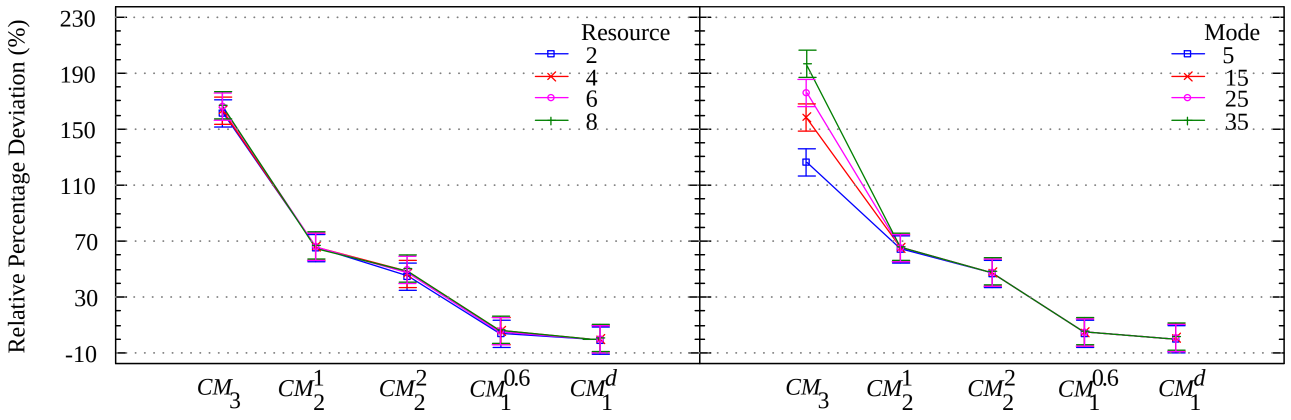


Fig. 13. Means plot and Tukey HSD intervals at the 95% confidence level for CM for instances with different number of resource types and modes.

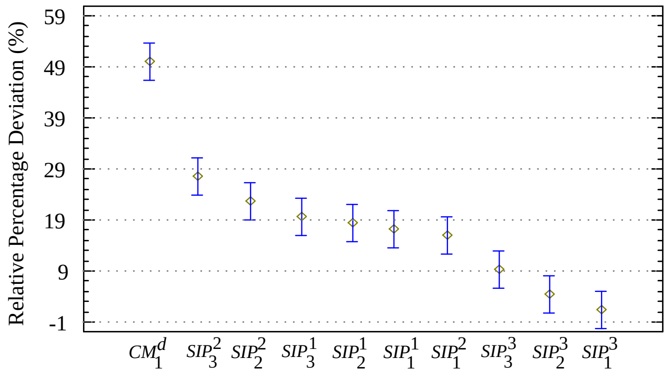


Fig. 14. Means plot and Tukey HSD intervals at the 95% confidence level of the schedule improvement procedures.

applications. During each test, the number of workflows are also set as $g \in \{50, 100, 200, 300, 400\}$. For each type of Workflow application, 50 tasks are generated using pegasus WorkflowGenerator. The start time and deadline of the workflow are set as the previous section.

Compared Algorithms: Methods for the related makespan based multiple workflow scheduling [5] are adapted for the considered problem. Three main strategies are adopted.

- Schedule the workflows one after another independently.
- Schedule parts of each workflow in turns independently.
- Combine the workflows into a big single workflow, and schedule the big workflow.

The proposed Path Cluster Heuristic (PCH) in [5] supposes that each task can only be processed on a single processor and uses the clustering technique to generate a group of tasks. So the PCH is not suitable for the mixed tasks in this paper. The Heterogeneous Earliest Finish Time (HEFT) [10] is a popular scheduling heuristic for minimizing the makespan of a single workflow. HEFT computes the rank value for each task which is the longest path from the current task to the dummy sink task. The task with the highest rank value is scheduled on the resource that gives the earliest finish time. HEFT is modified to the Iterative HEFT (IHEFT) for the renting cost minimization problem in this paper. All the resources are sorted in a non-descending order of their unit costs. Starting from the lower bound of each resource, HEFT is iteratively called to calculate the makespan of each workflow. If the deadlines of the workflows are not satisfied, the number of resources is increased one by one according to the non-descending order and HEFT is called again. The procedure stops when a feasible solution is found. The IHEFT is combined with the three multiple workflow scheduling strategies $IHEFT_i, i \in \{1, 2, 3\}$. The three adapted algorithms are compared with the best proposed algorithm SIP_1^3 in this section. Besides $IHEFT_i, i \in \{1, 2, 3\}$, the two lower bounds of the problem are also considered. $LB - o$ is the lower bound of the total renting cost with only on-demand resources. All the rented resources are fully used, which is cal-

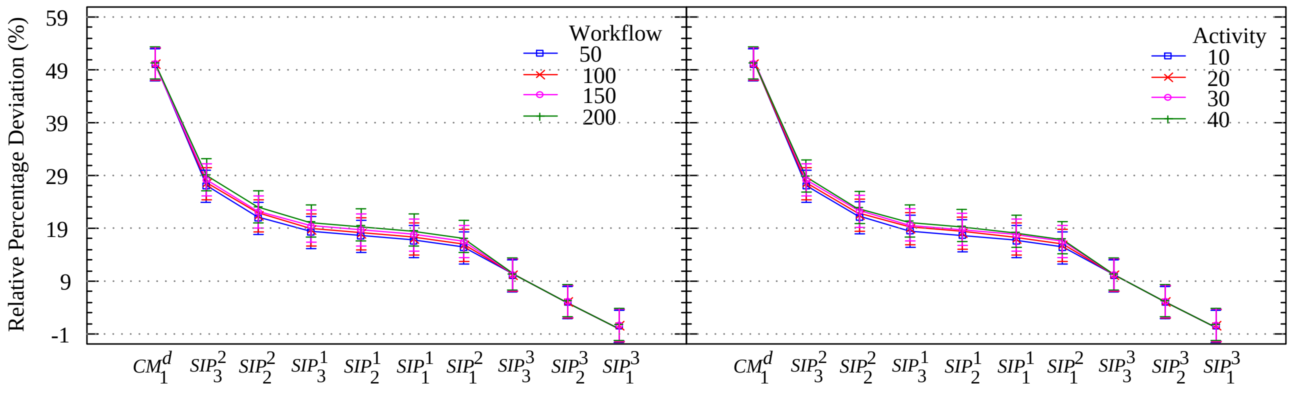


Fig. 15. Means plot and Tukey HSD intervals at the 95% confidence level for SIP for instances with different number of workflows and tasks.

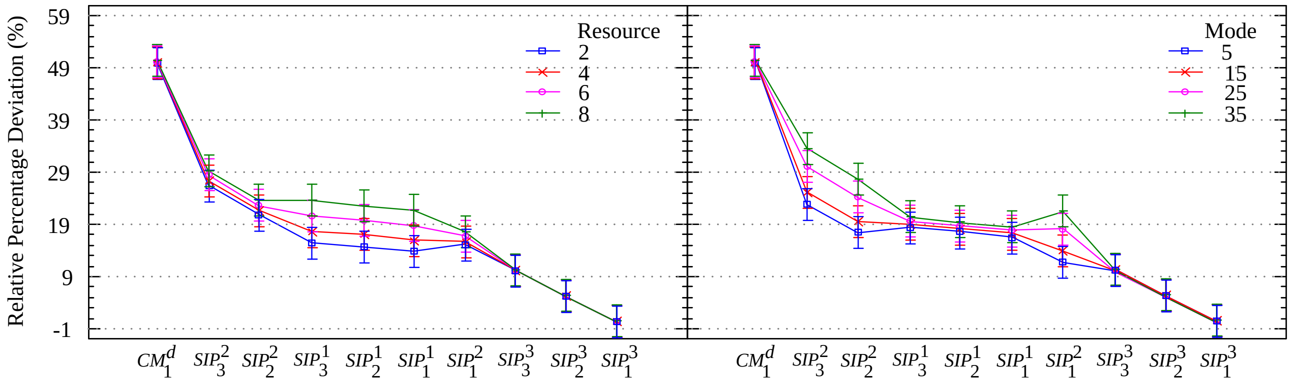


Fig. 16. Means plot and Tukey HSD intervals at the 95% confidence level for SIP for instances with different number of resource types and modes.

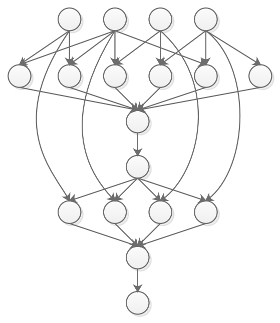


Fig. 17. An example of Montage workflow applications.

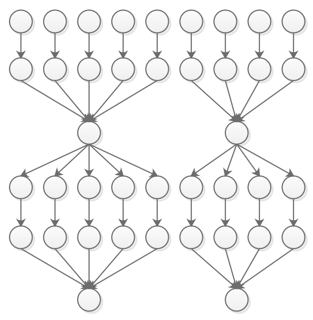


Fig. 18. An example of LIGO workflow applications.

culated as $\sum_{w=1}^{g}\sum_{j=1}^{n_w}\sum_{k=1}^{a}(c_k \min_{o=1}^{m^{jw}} r_{jok}^{w} d_{jo}^{w})/discount_k$ ($discount_k$ is the discount of virtual machine k obtained from Table 2). $LB-r$ is the lower bound with only reserved resources and is calculated as $\sum_{w=1}^{g}\sum_{j=1}^{n_w}\sum_{k=1}^{a}(c_k \min_{o=1}^{m^{jw}} r_{jok}^{w} d_{jo}^{w})$. The ANOVA technique is also used to analyze the results in a sound and statistical way where RPD is the response variable.

For Montage instances, the interaction plot between the number of workflows and the compared algorithms with 95% Tukey HSD confidence intervals is shown in Figure 19. For different workflow values g, $IHEFT_i, i \in \{1,2,3\}$ and SIP_1^3 are much better than $LB-o$ and just a little higher than $LB-r$. SIP_1^3 is better than the $IHEFT$ algorithms. The total renting cost of SIP_1^3 is about 40% higher than the lower bound with reserved instances and 200% lower than the lower bound with only on-demand instances. When the size g of the workflow is very small, the RPD of $IHEFT_i, i \in \{1,2,3\}$ and SIP_1^3 are almost the same. With the increase in g, the RPDs of $IHEFT$ increase fast while that of SIP_1^3 increases slowly. This implies that the proposed SIP_1^3 is much more suitable for the Montage applications in large instances.

For LIGO instances, the comparison results are shown in Figure 20. We can observe that SIP_1^3 also outperforms $IHEFT_i, i \in \{1,2,3\}$ and $LB-o$ for different values of g. The total renting cost of SIP_1^3 is about 20% higher than the lower bound with reserved instances and 220% lower than the lower bound with only on-demand instances. Regardless of the value of g, SIP_1^3 is much better than $IHEFT_i, i \in \{1,2,3\}$. With the increase in the instance size, SIP_1^3 shows the best performance. SIP_1^3 is suitable for LIGO applications of any size.

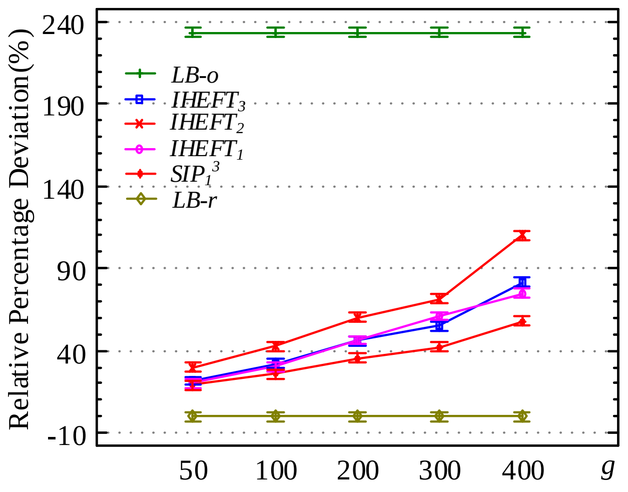


Fig. 19. Comparison results of the algorithms on Montage instances with different number of workflows.

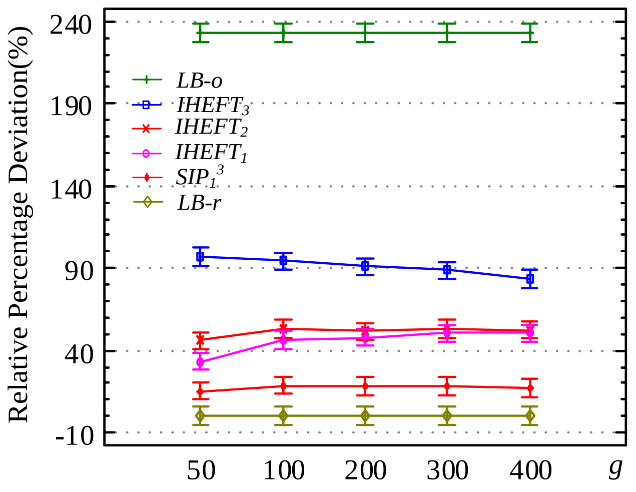


Fig. 20. Comparison results of the algorithms on LIGO instances with different number of workflows.

6 Conclusion and Future Work

In this paper the multiple periodical workflow scheduling problem with cost minimization is considered regarding the long-term resource rental relationship between the users and the CSP. The precedence tree based heuristic (PTH) is developed for the considered problem which contains a Synchronization based Workflows Combination procedure (SWC), three types of precedence tree based initial schedule Construction Methods (CM) and a mode and resource based Schedule Improvement Procedure (SIP). The proposed PTH is compared on randomly generated instances with algorithms for the similar makespan based multiple workflow scheduling problem. Experimental results demonstrate that the one-step heuristic with dynamic β is the best initial schedule construction method no matter the number of workflows, tasks, modes or types of resource. The SIP has a greater influence on the performance of the PTH. The performance of MMPE increases with the rising in the number of execution modes while RAP increases with the rising in the types of resource. The combination of MMPE and RAP gives the best result when comparing to other adapted algorithms. The proposed PTH is also compared with the adapted algorithms on a simulated real public cloud. PTH saves a lot of cost for both Montage and LIGO applications while comparing with resource provisioning with only on-demand instances.

Few works have been carried out on the multiple workflows scheduling problem. The proposed schedule construction and improvement procedures can be easily adapted to other workflow based scheduling problems. Future work might include more accurate models to predict the arrival of workflows. Problems with resources that are not sharable between workflows are also worth considering.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 61572127, 61272377), the Key Research & Development program in Jiangsu Province (No. BE2015728) and Collaborative Innovation Center of Wireless Communications Technology. 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] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-science: An overview of workflow system features and capabilities," *Future Generation Computer Systems*, vol. 25, no. 5, pp. 528–540, 2009.

[2] D. A. Brown, P. R. Brady, A. Dietz, J. Cao, B. Johnson, and J. McNabb, "A case study on the use of workflow technologies for scientific analysis: Gravitational wave data analysis," in *Workflows for e-Science*. Springer, 2007, pp. 39–59.

[3] AmazonEC2, "Amazon elastic compute cloud (Amazon EC2)," *http://aws.amazon.com/ec2/pricing*, 2014.

[4] M. Xu, L. Cui, H. Wang, and Y. Bi, "A multiple QoS constrained scheduling strategy of multiple workflows for cloud computing," in *IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA 2009)*. IEEE, 2009, pp. 629–634.

[5] L. F. Bittencourt and E. R. Madeira, "Towards the scheduling of multiple workflows on computational grids," *Journal of grid computing*, vol. 8, no. 3, pp. 419–441, 2010.

[6] J. Yu and R. Buyya, "Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms," *Scientific Programming*, vol. 14, no. 3, pp. 217–230, 2006.

[7] E. Demeulemeester, W. S. Herroelen, and S. E. Elmaghraby, "Optimal procedures for the discrete time/cost trade-off problem in project networks," *European Journal of Operational Research*, vol. 88, no. 1, pp. 50–68, 1996.

[8] E. Demeulemeester, B. De Reyck, B. Foubert, W. S. Herroelen, and M. Vanhoucke, "New computational results on the discrete time/cost trade-off problem in project networks," *Journal of the Operational Research Society*, vol. 49, no. 11, pp. 1153–1163, 1998.

[9] Ö. Hazır, M. Haouari, and E. Erel, "Discrete time/cost trade-off problem: A decomposition-based solution algorithm for the budget version," *Computers & Operations Research*, vol. 37, no. 4, pp. 649–655, 2010.

[10] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[11] A. Radulescu and A. J. Van Gemund, "A low-cost approach towards mixed task and data parallel scheduling," in *International Conference on Parallel Processing (ICPP2001)*. IEEE, 2001, pp. 69–76.

[12] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and 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 and 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*, vol. 39, no. 1, pp. 29–43, 2009.

[14] J. Yu, R. Buyya, and 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, and X. Zhu, "Deadline division-based heuristic for cost optimization in workflow scheduling," *Information Sciences*, vol. 179, no. 15, pp. 2562–2575, 2009.

[16] S. Abrishami, M. Naghibzadeh, and D. Epema, "Cost-driven scheduling of grid workflows using partial critical paths," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 8, pp. 1400–1414, 2012.

[17] Z. Cai, X. Li, and 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 and 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] Z. J. Wu, X. Liu, Z. W. Ni, D. Yuan, and Y. Yang, "A market-oriented hierarchical scheduling strategy in cloud workflow systems," *Journal of Supercomputing*, vol. 63, no. 1, pp. 256–293, 2013.

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

[21] R. Prodan, M. Wieczorek, and H. M. Fard, "Double auction-based scheduling of scientific applications in distributed grid and cloud environments," *Journal of Grid Computing*, vol. 9, no. 4, pp. 531–548, 2011.

[22] Y. C. Lee, R. Subrata, and 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*, vol. 20, no. 9, pp. 1273–1284, 2009.

[23] H. M. Fard, R. Prodan, and T. Fahringer, "Multi-objective list scheduling of workflow applications in distributed computing infrastructures," *Journal of Parallel and Distributed Computing*, vol. 74, no. 3, pp. 2152–2165, 2014.

[24] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *IEEE Transactions on Parallel and Distributed systems*, vol. 22, no. 6, pp. 931–945, 2011.

[25] E. K. Byun, Y. S. Kee, J. S. Kim, E. Deelman, and S. Maeng, "BTS: Resource capacity estimate for time-targeted science workflows," *Journal of Parallel and Distributed Computing*, vol. 71, no. 6, pp. 848–862, 2011.

[26] E. K. Byun, Y. S. Kee, J. S. Kim, and S. Maeng, "Cost optimized provisioning of elastic resources for application workflows," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1011–1026, 2011.

[27] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, 2013.

[28] Z. Cai, X. Li, and J. N. D. Gupta, "Heuristics for provisioning services to workflows in XaaS clouds," *IEEE Transactions on Services Computing*, vol. DOI10.1109/TSC.2014.2361320, 2014.

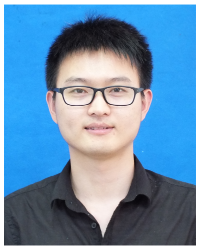
[29] S. Chaisiri, B. S. Lee, and D. Niyato, "Optimization of resource provisioning cost in cloud computing," *IEEE Transactions on Services Computing*, vol. 5, no. 2, pp. 164–177, 2012.

[30] X. Zuo, G. Zhang, and W. Tan, "Self-adaptive learning pso-based deadline constrained task scheduling for hybrid iaas cloud," *IEEE Transactions on Automation Science and Engineering*, 2013.

[31] E. Demeulemeester and W. S. Herroelen, *Project scheduling: a research handbook*. Kluwer Academic Pub, 2002, vol. 49.

[32] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[33] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Characterization of Scientific Workflows," in *2008 THIRD WORKSHOP ON WORKFLOWS IN SUPPORT OF LARGE-SCALE SCIENCE (WORKS 2008)*, 2008, pp. 11–20, 3rd Workshop on Workflows in Support of Large-Scale Science, Austin, TX, NOV 17, 2008.



Long Chen received his B.Sc. degrees in Computer Science and Engineering from Southeast University, Nanjing, China, in 2009. He is currently a Ph.D. student at the School of Computer Science and Engineering, Southeast University, Nanjing, China. He is a student member of the IEEE. His main interests focus on Dynamic Capacity Management, Task Scheduling in Cloud Computing, Service-oriented Computing, Evolutionary Computation, Project Scheduling.

Xiaoping Li (M09-SM12) received his B.Sc. and M.Sc. degrees in Applied Computer Science from the Harbin University of Science and Technology, Harbin, China, in 1993 and 1999 respectively. He obtained his Ph.D. degree in Applied Computer Science from the Harbin Institute of Technology, Harbin, China, in 2002. He joined Southeast University, Nanjing, China, in 2005, and is currently a professor at the School of Computer Science and Engineering. From Jan. 2003 to Dec. 2004, he did postdoctoral research at the Department of Automation at Tsinghua University, Beijing, China. From Mar. 2009 to Mar. 2010, he was a visiting professor at the National Research Council, London, Ontario, Canada. He is the author or co-author over more than 100 academic papers, some of which have been published in international journals such as *IEEE Transactions on Services Computing*, *IEEE Transactions on Automation Science and Engineering*, *Omega*, *European Journal of Operational Research*, *Information Sciences*, *International Journal of Production Research*, *Expert Systems with Applications*, *Journal of Network and Computer Applications* and *Engineering Optimization*. His research interests focus on Scheduling in Cloud Computing, Scheduling in Cloud Manufacturing, Machine Scheduling, Project Scheduling, Terminal Container Scheduling, Learning Effects in Scheduling, and Manufacturing Software Interoperability.

Rubén Ruiz is full professor of Statistics and Operations Research at the Polytechnic University of Valencia, Spain. He is co-author of more than 60 papers in International Journals and has participated in presentations of more than a hundred papers in national and international conferences. He is editor of the Elseviers journal *Operations Research Perspectives (ORP)* and co-editor of the JCR-listed journal *European Journal of Industrial Engineering (EJIE)*. He is also associate editor of other important journals like *TOP* or *Applied Mathematics and Computation* as well as member of the editorial boards of several journals most notably *European Journal of Operational Research* and *Computers and Operations Research*. He is the director of the Applied Optimization Systems Group (SOA, http://soa.iti.es) at the Instituto Tecnolgico de Informtica (ITI, http://www.iti.es) where he has been principal investigator of several public research projects as well as privately funded projects with industrial companies. His research interests include scheduling and routing in real life scenarios.