Hybrid resource provisioning for cloud workflows with malleable and rigid tasks

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

***Abstract*—In cloud computing, reserved and on-demand instances are generally provided by service providers. Hybridization of the two alternatives can considerably save costs when renting resources from the cloud. However, it is a big challenge to determine the appropriate amount of reserved and on-demand resources in terms of users' requirements. In this paper, the workflow scheduling problem with both reserved and on-demand instances is considered. The objective is to minimize the total rental cost under deadline constrains. The considered problem is mathematically modeled. A multiple sequence-based earliest finish time method is proposed to construct schedules for the workflows. Four different rules are used to generate initial task allocation sequences. Types and quantities of resources are determined by a free time block-based schedule construction mechanism. New sequences are generated by a variable neighborhood search method. Experimental and statistical analyses and results demonstrate that the proposed algorithm algorithm generates considerable cost savings when compared to the algorithms with only on-demand or reserved instances.**

***Index Terms*—Workflow scheduling, Cloud computing, Hybrid resource provisioning, Malleable task.**

I. Introduction

COMPLEX workflow applications are widespread in scientific and business analysis, e.g., astronomy (Montage, LIGO), earthquake detection (CyberShake) and genome sequencing (Epigeomics, SIPHT) [1]. They usually are executed on distributed or cloud systems, such as Pegasus [2], Askalon [3], Google MapReduce [4] or Amazon EC2 [5]. High quality computing and storage resources (networks, servers, storage, applications and services, etc.) are provided by Cloud Service Providers (CSPs) for workflow applications [6]. Generally, there are two types of tasks in workflow applications: rigid and malleable. Rigid tasks are executed only on given number of VMs all of the same type. A malleable task has multiple execution modes which can be executed in parallel on several VMs. Take the scientific workflow application Montage for example, as shown in Figure 1, mProjectPP, mDiffFit and mBcakground are malleable tasks while tasks on the other nodes are rigid. Though malleable tasks have been studied in many existing works [7][8][9], little attention has been paid to rigid tasks. However, rigid tasks are most of the time a bottleneck for the whole workflow application. In other words, both malleable and rigid tasks are crucial for the scheduling performance of workflow applications. In addition, users rent resources from CSPs without acquisition and maintenance costs. CSPs always provide two types of resource renting alternatives: reserved and on-demand [10]. Users can utilize reserved resources with one-time payment for a relatively long time. The average cost is much lower as significant discounts can be received for long time renting. Since resource requirements cannot be precisely predicted, short-term resource renting is necessary for peak requirements. The average unit cost of the on-demand instances is usually higher than that of the reserved one. Most existing studies mainly focus on single resource provisioning for workflow applications: reserved alternative [11][12] or on-demand alternative [8][13].

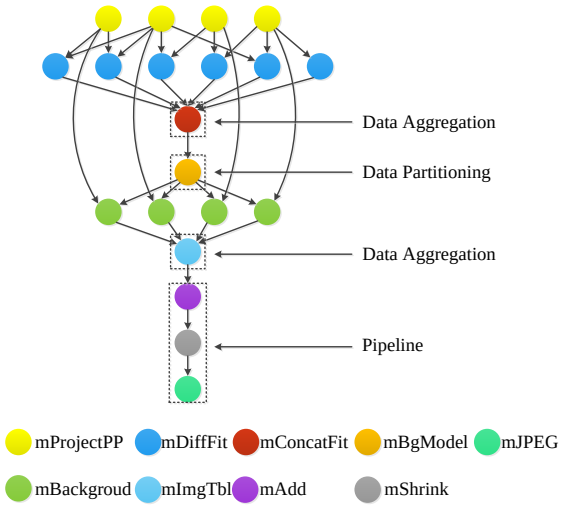


Fig. 1. An example for the Montage application with rigid and malleable tasks.

In this paper, we consider the problem of scheduling tasks of a workflow application to VMs in cloud computing to minimize the total rental cost. Both malleable and rigid tasks are considered. VMs are rented by the on-demand or reserved alternatives. Each workflow application is constrained by a deadline given in advance. Hybrid task types and different resource renting alternatives make the complex precedence constrained workflow scheduling problem difficult to solve. The main challenges are: (i) Multi-mode malleable tasks make the determination of their execution times complex. The critical path of the workflow cannot be verified in advance and it is difficult to find the optimum solution for the considered problem. (ii) The complex workflow structure and uncertain

Long Chen, Xiaoping Li and Yucheng Guo 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: 86-25-52091916; Fax: 86-25- 52091916; 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 .

critical path make it hard to estimate the amount of reserved and on-demand resources to which the objective (the total rental cost) is closely related. (iii) Due to the deadline of the workflow application and unpredicted processing times of malleable tasks, it is complicated to determine the amount of rented on-demand resources in each time window.

We propose a multiple sequence-based earliest finish time method (MEFT) for the problem under study. Compared to our previous work [14], a new resource and sequence initialization method, schedule construction method and schedule improvement method are proposed. The proposed MEFT has three main benefits. First, MEFT generates multiple task allocation sequences for workflow applications with malleable and rigid tasks. Second, the free time block-based schedule construction method determines the amount and renting alternatives of virtual machines and allocates each task to a suitable virtual machine. Third, the backward and forward movement based improvement method balances the usage of resource and reduces the amount of on-demand resources. Parameters and components of the proposed algorithm are calibrated and analyzed over a number of instances using the Design of Experiments approach and a multi-factor analysis of variance statistical technique. The proposal is compared with existing state-of-the-art algorithms for related problems. Thorough experiments demonstrate the superiority of the proposed approach.

The rest of the paper is organized as follows. Section II describes the existing literature. The definition and the mathematical model for the considered problem are given in Section III. In Section IV, the proposed MEFT algorithm is described. Computational results are shown in Section V followed by conclusions and future avenues of research in Section VI.

II. Related Works

Due to its relevance, the problem of scheduling workflow applications has been a hot topic for research over recent years. In the traditional distributed computation field (utility Grids), resources are encapsulated as services and services are not shareable between workflow tasks. There are two main objectives: cost optimization under deadline constraints and execution time optimization under budget constraints [15]. Common methods for time optimization include dynamic programming [16], branch and bound [17], decomposition-based methods [18], list scheduling [19], critical path based allocation [20], greedy randomized adaptive search [21] and ant colony optimization approach [22]. The methods for cost optimization include the deadline-MDP algorithm [23], DET (Deadline Early Tree) algorithm [24], PCP (Partial Critical Paths) algorithm [7] and the CPI (Critical Path-based Iterative) heuristic [25]. These are just samples of the most relevant works.

In cloud environments, resources are assumed to be unlimited and continuously available. Huang et al. [26] proposed a mechanism to determine the minimum set of resources to minimize the makespan of a workflow. The set size relied on the scale of the DAG, communication costs between tasks, computation costs and degrees of parallelism and uniformity of the tasks, etc. Though the mechanism obtained good results for the given sample workflows, efficiency was not guaranteed for actual workflows with complex parallel degrees and uniformities. Byun et al. [11] 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 follow up work [27], the Partitioned Balanced Time Scheduling (PBTS) algorithm was presented for homogeneous resources in which resources are provided with an on-demand option. PBTS considers time partitions in the algorithm and minimizes the amount of resources for each time partition. Abrishami et al. [7] 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 [8] in a cloud environment. The two algorithms IC-PCP and IC-PCPD2 are proposed. Unlike utility grids they contain on demand on-demand resource provisioning, homogeneous networks, and the pay-as-you-go pricing model. Followed by Cai et al. [9], the workflow scheduling problem with heterogeneous resources and on-demand resource provisioning was considered. Two heuristics CPIS and LHCM were proposed to solve the two sub-problems which were referred to as service mapping and task tabling. However, these papers considered scheduling workflows with only on-demand or reserved resources. Scheduling algorithms including both on-demand and reserved resources were not considered.

There are a few studies on both the reserved and on-demand resource provisioning strategies. Chaisiri et al. [13] proposed an optimal cloud resource provisioning (OCRP) algorithm to get a balance between the reserved and the on-demand. A stochastic programming model was formulated. The demand distribution of resources at each decision stage is supposed to be known in advance. Polynomial heuristics were proposed by Khatua et al. [28] for the hybrid resource provisioning problem. Van den Bossche et al. [29] presented a purchase management algorithm to automate procurement decisions on reserved contracts in the contexts of providers. However, these studies actually focused on independent task scheduling, not on DAG workflows.

To the best of our knowledge, there is no existing work considering both malleable tasks and hybrid resource provisioning for workflow scheduling subject to deadlines. However, workflow applications with malleable and rigid tasks are commonly exist in scientific computing. Hybrid resource provisioning can significantly reduce the total renting cost for workflow applications. Therefore, this paper considers the workflow scheduling problem with hybrid resource provisioning methods.

III. Problem Description and Formulation

A. Problem Description

The task-on-node Directed Acyclic Graph (DAG) is commonly used to represent a workflow application in which tasks are denoted by nodes and dependencies between tasks are represented by edges. Dependencies between tasks are

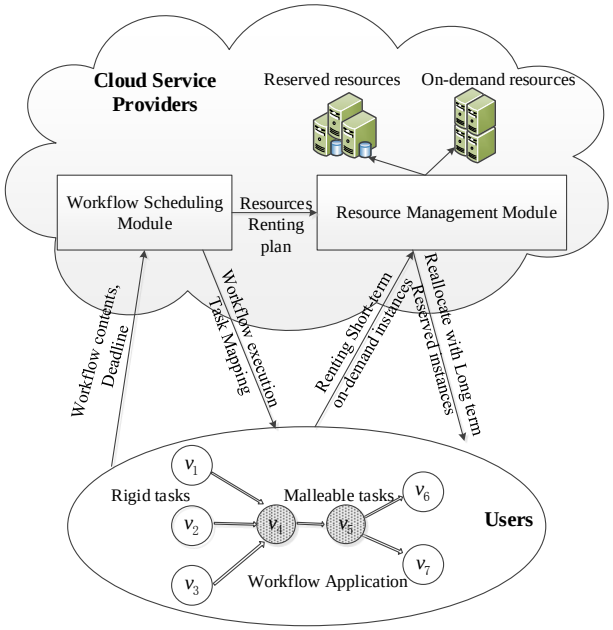


Fig. 2. Hybrid resource provisioning for the considered cloud workflow.

represented by edges. Each task can be processed on one or several virtual machines. We only consider homogeneous virtual machines in this paper. Physical machines with different configurations are virtualized to the same virtual machine type with identical CPU cores. Resources are available for all tasks, i.e., each task can be processed on any virtual machine. Resources are used by other tasks after the current task finishes during the resource rental period.

Figure 2 shows the proposed workflow scheduling framework. Users send their requests to CSPs. These requests are represented by workflow applications. Parameters of workflows (deadlines, tasks and runtimes) are distributed to the workflow scheduling module. The module generates scheduling plans according to these parameters. Each scheduling plan includes two parts: the workflow schedule and the resource rental plan. The workflow schedule contains the start time of each task and the resource rental plan contains the quantities and the resource renting alternatives. The resource rental plan is sent to the the resource management module. The resource management module rents proper on-demand and reserved resources for different tasks based on the rental plan. The workflow schedule and the rented resources are than sent back to the users.

B. Mathematical Model

Let the workflow be $G(V, E)$. $V = \{v_0, \ldots, v_n\}$ and $E = \{(v_i, v_j) | v_i \in V, v_j \in V, i < j\}$ define the set of tasks and edges in G. The two dummy nodes v_0 and v_n represent the start and the end of the workflow. Each node v_i can be processed on several Virtual Machines (VM). Let the processing time of task v_i on a single virtual machine be P_i. We use the set $\mathbb{M}$ and $\mathbb{R}$ to denote malleable and rigid tasks, i.e., $V = \mathbb{M} \cup \mathbb{R}$.

If $v_i \in \mathbb{M}$, then it is a malleable task and processed on m_i virtual machines, the final processing time p_i is denoted by $p_i = \lceil P_i/m_i \rceil$, which is determined by the currently available virtual machines during the scheduling process. However, for a rigid task v_i ($v_i \in \mathbb{R}$), it can only be processed on a fixed number of virtual machines. The processing time $p_i = \lceil P_i/m_i \rceil$ is unchanged during the scheduling process. p_i is supposed to be integer in this paper since resources are commonly charged for per hour in cloud environments. Let s_i and f_i be the start and finish times of task v_i. The deadline of the entire workflow is given by D.

During the scheduling process, the total number of rented resources is H. Let H^r be the amount of reserved resources with a unit cost of C_r. Once a reserved resource is used, it will not be released until the whole workflow application is finished (the dummy end task finishes at time f_n). Let H_t^0 be the amount of on-demand resources at time t with a unit cost of C_0. The on-demand resources will be released once all tasks executed on the resource are finished. The discount is denoted as the ratio of C_r to C_o, i.e., $discount = C_r/C_0$. Then, the considered problem can be modeled as follows. The objective is

$$\min(f_n \times H^r \times C_r + \sum_{t=0}^{f_n} H_t^0 \times C_0) \tag{1}$$

Two sets of binary variables are defined. Binary variables x_{iht} in Equation (2) take value 1 if and only if task v_i is executed on VM h at time t and it is 0 otherwise. Binary variables y_h in Equation (3) take value 1 if and only if the VM h is an on-demand instance and it is 0 otherwise.

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

$$y_h = \begin{cases} 1 \;\; if \; h \text{ is an on-demand virtual machine}, \\ \forall h \in \{1, \ldots, H\} \\ 0 \;\; if \; h \text{ is a reserved virtual machine}, \\ \forall h \in \{1, \ldots, H\} \end{cases} \tag{3}$$

subject to the following constraints:

$$m_i = \sum_{h=1}^{H} \sum_{t=0}^{D} x_{iht}, \; \forall v_i \in \mathbb{M} \tag{4}$$

$$p_i = \lceil P_i/m_i \rceil, \quad \forall i \in \{0, \ldots, n\} \tag{5}$$

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

$$f_i = s_i + p_i, \; \forall i \in \{0, \ldots, n\} \tag{7}$$

$$s_i + p_i \leqslant s_j, \; \forall (i, j) \in E \tag{8}$$

$$f_n \leqslant D \tag{9}$$

$$\sum_{t=0}^{D} x_{iht} = 1, \quad \forall i \in \{0, \ldots, n\}, \; \forall h \in \{1, \ldots, H\} \tag{10}$$

$$H^0_{t_i} = \sum_{h=1}^{H} \sum_{i=0}^{n} \sum_{t=t_i-p_i+1}^{t_i} x_{iht} \times y_h, \quad \forall t_i \in \{0, \ldots, D\} \quad (11)$$

$$H^r = \sum_{h=1}^{H} (1 - y_h) \quad (12)$$

The number of VMs for malleable tasks is controlled by Equation (4). Equations (5), (6) and (7) calculate the processing, start and finish times of task v_i. Workflow constraints are specified in constraint set (8) and (9). Equation (10) ensures that each task can only start at one time, considering that the execution of a task is non-preemptive. Finally, Equations (11) and (12) give the amount of reserved and on-demand resources in the schedule.

C. An Example

Take the workflow in Figure 2 as an example. Parameters of the workflow are given in Table I. There are seven tasks denoted as v_i $(i = 1, \ldots, 7)$ and m denotes the amount of resources needed. The runtime (the processing time on a single virtual machine) of each task is 3, 1, 2, 2, 2, 2, 2 respectively. v_3 and v_4 are malleable tasks with changeable amounts of resources and variable processing times, e.g., v_3 can be executed on 1 virtual machine with a processing time of 2 or on 2 virtual machines with a processing time of 1. The execution mode of v_3 is determined by its runtime and the virtual machines currently available. The remaining tasks are rigid tasks requiring fixed resources in which v_5 and v_6 require two virtual machines while the other rigid tasks require only one virtual machine. v_5 is executed on 2 virtual machines with a processing time of 1, which cannot be changed. The actual processing time of v_i is calculated by $p_i = \lceil Runtime/m \rceil$. Actual processing times of the tasks in the example are 3, 1, 2, 2, 1, 1, 2 respectively. $discount$ denotes the ratio of the average unit cost of reserved resources to that of on-demand ones. Let the deadline of the example workflow be 9 and the $discount$ be 0.8.

TABLE I
PARAMETERS OF THE WORKFLOW EXAMPLE.

Task	Runtime	Type	m
1	3	Rigid	1
2	1	Rigid	1
3	2	Malleable	1
4	2	Malleable	1
5	2	Rigid	2
6	2	Rigid	2
7	2	Rigid	1

Figure 3 shows a schedule of the above example where each task finishes as early as possible. Tasks v_1, v_2 and v_3 start at the same time in parallel. Two virtual machines are allocated to malleable tasks v_3 and v_4, and their processing times become 1. If the virtual machines are all rented using the reserved method, the total renting cost is $4 \times 7 \times 0.8 = 22.4$. Since the resource utilization rate of virtual machine 1 is high, if only virtual machine 1 is reserved, the total renting cost is $6 \times 0.8 + 4 + 3 + 1 = 12.8$. If both virtual machine 1 and virtual machine 2 are reserved, the renting cost is $6 \times 0.8 + 6 \times 0.8 + 3 + 1 = 13.6$. As can be seen, the differences in costs are highly significant.

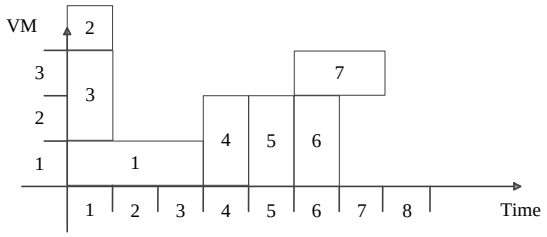


Fig. 3. A schedule of the example with tasks finishing as early as possible.

Cloud workflows can be scheduled more effectively by the hybrid resource provisioning strategy, i.e., reserved resources can save costs if high utilization is achieved, while on-demand resources can improve resource utilization during peak demand periods. For example, Figure 4 shows the optimal schedule of Figure 4. v_3 is allocated to one virtual machine and v_4 to two virtual machines. Virtual machines 1 and 2 are rented using the reserved strategy while virtual machine 3 is rented using the on-demand alternative only in time slot 7. The total renting cost is $7 \times 0.8 + 7 \times 0.8 + 1 = 12.2$.

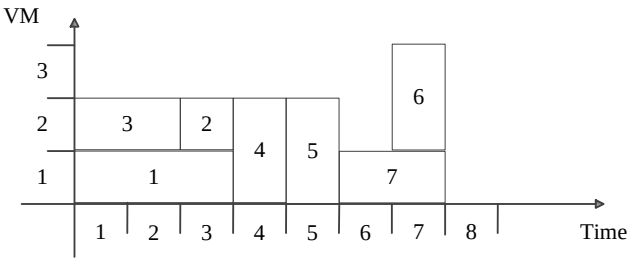


Fig. 4. The optimal schedule of the example with hybrid resources renting.

IV. MULTIPLE SEQUENCE-BASED EARLIEST FINISH TIME HEURISTIC

As the considered problem is NP-hard, exact algorithms are viable only in small instances. Rule-based heuristics are proposed for the considered workflow scheduling problems. The proposed multiple sequence-based earliest finish time method (MEFT) is composed of five components: resource and sequence initialization (RSI), schedule construction (SC), schedule improvement (SI), schedule reconstruction (SR) and multiple sequence generation (MSG). In the RSI, the amount of reserved resources is initialized according to some lower and upper bounds. The initial task allocation sequence is determined according to the priorities of each task. The procedure SC schedules tasks sequentially based on the task allocation sequence. After obtaining a schedule, malleable tasks are readjusted to improve the schedule by using the SI procedure. In the schedule reconstruction (SR) phase, some tasks with on-demand resources are randomly selected and changed to reserved resources. Finally, the variable neighborhood search method is used to improve the task allocation sequences in the MSG step. An outline of the MEFT procedure is shown

in Algorithm 1. All these steps and components are detailed in the following sections.

Algorithm 1: multiple sequence-based earliest finish time method (MEFT)

```
1 begin
2     Resource and sequence initialization (RSI);
3     repeat
4         Schedule construction (SC);
5         Schedule improvement (SI);
6         Schedule reconstruction (SR);
7         multiple sequences generation (MSG);
8     until (Termination criterion is satisfied);
9     return;
```

A. Resource and sequence initialization (RSI)

There are two initialization phases in RSI: reserved resource initialization (RRI) and task allocation sequence initialization (TASI). The reserved resource initialization phase determines the initial amount of reserved resources while the task allocation sequence gives the initial order of tasks.

1) Reserved resource initialization (RRI): The reserved resource initialization process avoids unnecessary attempts at finding a suitable amount of reserved resources during the schedule construction process. The upper and lower bounds of the reserved resources are determined first. To calculate the lower bound, the precedence constraints between tasks are not considered and all the resources are assumed to be fully used. Only the deadline constraints are considered. The minimum amount of resources required to accomplish all the tasks is denoted as the lower bound, which is calculated as follows:

$$H^r_{\min} = \lceil \sum_{v_i \in V} P_i/D \rceil \tag{13}$$

According to Formula (1), if a virtual machine is reserved, the renting cost is $C_r \times f_n$. If a virtual machine is on-demand, the renting cost is $\sum_{t=0}^{f_n} H^0_t \times C_0$ ($H_t = 1$ if this on-demand virtual machine is used at time t, otherwise is $H_t = 0$). Therefore, if the utilization rate $((\sum_{t=0}^{f_n} H^0_t \times C_0)/(C_r \times f_n))$ is less than the $discount$ (C_0/C_r), it is appropriate to use the on-demand strategy to rent a resource. Therefore, if the resource utilization equals $discount$, the amount of resources required is denoted as the upper bound which is calculated as follows:

$$H^r_{\max} = \lceil \sum_{v_i \in V} P_i/D/discount \rceil \tag{14}$$

To determine the initial H^r, the reserved resource initialization process starts from $H^r = H^r_{\min}$. The feasibility of the current H^r is checked by allocating each malleable task to the maximum amount of resources, i.e., all tasks are assumed to finish as early as possible. For all $v_i \in \mathbb{M}$, the maximum amount of resources m_i considering and not considering $discount$ are $m_i = H^r/(1 - discount)$ and $m_i = H^r$ respectively. The earliest and latest start times est_i and lst_i of v_i are calculated with the critical path based dynamic programming method proposed by [30]. If $est_n > D$, the current H^r is unfeasible and no feasible schedule can be found. H^r is increased by one and the above process is iterated until a feasible schedule is obtained or $H^r = H^r_{\max}$. Algorithm 2 shows the details of the reserved resource initialization method. The time complexity for finding a suitable H^r in step 4 is $O(n)$, the time complexity to calculate est_n in step 7 is $O(n^2)$, so the time complexity of RSI is $O(n^3)$ to obtain the initial amount of reserved resources.

Algorithm 2: Reservation resource initialization (RSI)

```
1  begin
2      H^r_min ← ⌈∑_{v_i∈V} P_i/D⌉;
3      H^r_max ← ⌈∑_{v_i∈V} P_i/D discount⌉;
4      for (H^r = H^r_min; H^r ≤ H^r_max; H^r ← H^r + 1) do
5          for each v_i ∈ V do
6              if v_i ∈ M then
7                  p_i ← ⌈P_i/(H^r/(1 − discount))⌉;
8          Calculate est_n;
9          if est_n > D then
10             continue;
11     return H^r;
```

2) Task allocation sequence initialization (TASI): After obtaining the amount of reserved resources H^r, tasks are sequenced by their priorities, which are determined by the upward-rank values [19]. The upward-rank value based priority considers both the structure of the workflow and the characteristics of the tasks. Suppose the upward-rank value of the dummy task v_n is $rank_u(v_n) = 0$. The upward-rank values are recursively calculated by $rank_u(v_i) = P_i + \max_{v_j \in succ(v_i)}\{rank_u(v_j)\}$ in which $succ(v_i)$ denotes the successors of a task v_i. If the processing time of a task is longer, it has a higher priority. Therefore, tasks on the critical path will get a higher priority and will be processed first. The task allocation sequence is initialized by sorting tasks according to their priorities.

Besides the maximum upward-rank value rule, three other different task sequencing rules are proposed:

1) minimal processing time: In this rule, only the characteristics of the task are considered. The task's priorities are determined by the processing time and the minimal processing time has the highest priority.
2) 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.
3) 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.

B. Schedule construction (SC)

In the SC, the free time block-based schedule construction procedure is used to schedule tasks. In traditional workflow

scheduling, the task sequence is a topological task order, i.e., all precedence constraints are met in the task allocation sequence. If task v_i is a predecessor of v_j, v_i is placed in front of v_j in the task allocation sequence. A task can only be scheduled after all of its predecessors have been scheduled. In the free time block-based schedule construction strategy, tasks are inserted into the available free time blocks. The earliest start time is determined by the earliest available free time blocks, which are updated at each step of the task scheduling process.

Matrix $R = (r_{ij})_{D\times H}$ denotes the resources. Rows and columns in matrix R represent time slots and virtual machines respectively. For example, $r_{ij} = 1$ means the virtual machine j at time slot i is unavailable. Therefore, if a task is processed, the elements in the corresponding sub-matrix are set to 1. For example, if the elements in the sub-matrix $R[i^{'}, \ldots, i^{''}; j^{'}, \ldots, j^{''}]$ are all 1, it means that the task is executed on virtual machine $j^{'}$ to $j^{''}$ from time slot $i^{'}$ to $i^{''}$. If all the elements in a sub-matrix are 0, we call this sub-matrix a free time block. Free time blocks are continuous time periods in which a number of virtual machines are available. For a task, the free time block is between its earliest start time and latest finish time.

Tasks are allocated to free time blocks as early as possible. First, all the free time blocks are sorted according to their start times. For a rigid task $v_i \in \mathbb{R}$, the first free time block is checked. There is no sufficient time for v_i if the duration of the block is less than the task processing time p_i. The current time block is unavailable and the next time block is checked. If there are no sufficient resources for v_i, we consider renting the least amount of on-demand resources. If the current free time block is not allocated to v_i, we calculate the wasted workload of the current task. Let the workload of v_i be w_i and the wasted workload of the free time block be Φ. If $\Phi \times discount > (w_i - \Phi)$, some on-demand resources are rented. v_i can be allocated to the current free time block. For a malleable task $v_i \in \mathbb{M}$, the allocation process is much more complex as the amount of resources can change. If the current time block is available for v_i, v_i is mapped to the maximum amount of virtual machines in order to finish as early as possible. Otherwise, we rent $m_i = H^r/(1 - discount)$ amount of on-demand resources. After a task is scheduled, the earliest start times of its immediate successors are updated. Details of the SC procurement are described in Algorithm 3. The time complexity of SC is $O(n^3)$ to construct the complete schedule for the problem.

Algorithm 3: Schedule construction (SC)

Input: the resource matrix $(r_{ij})_{D\times H}$, the amount of reserved resources H^r, the task allocation sequence $\bar{s}$.
Output: the current schedule π.
begin
  $idleList \leftarrow \varnothing$;
  for *($i = est_k$;$i < t_{last}$;$i \leftarrow i + 1$)* **do**
    for *($j = 0$;$j < H$;$j \leftarrow j + 1$)* **do**
      if *($r_{ij} = 0$)* **then**
        Check all free time blocks starting from r_{ij};
        Add the free time block to $idleList$;
  Combine time slots after t_{last} to create a free time block;
  Add the free time block to $idleList$;
  repeat
    $v_i \leftarrow$ the first task of $\bar{s}$;
    if $v_i \in \mathbb{R}$ **then**
      for *($j = 0$;$j < |idleList|$;$j \leftarrow j + 1$)* **do**
        if *the time of $idleList_j$ is not available for v_i* **then**
          continue;
        if *the resources of $idleList_j$ are not available for v_i* **then**
          $\Phi \leftarrow$ the workload $idleList_j$, $w_i \leftarrow$ the workload of v_i ;
          if $\Phi \times discount > w_i - \Phi$ **then**
            Rent new on-demand instances for v_i;
            Allocate v_i to $idleList_j$;
          else
            continue;
        if *$idleList_j$ is available for v_i* **then**
          Allocate v_i to $idleList_j$;
    if $v_i \in \mathbb{M}$ **then**
      for *($j = 0$;$j < |idleList|$;$j \leftarrow j + 1$)* **do**
        if *$idleList_j$ is available for v_i* **then**
          Allocate v_i to the maximum amount resources of $idleList_j$;
        if *$idleList_j$ is not available for v_i* **then**
          $m_i \leftarrow H_r/(1 - discount)$;
          Rent m_i amount of on-demand instances for v_i;
          Allocate v_i to $idleList_j$;
    Update $idleList$;
    Remove the task of $\bar{s}$;
  until *($\bar{s} = \varnothing$)*;
  return π;

C. Schedule improvement (SI)

After the schedule construction process, the following time periods are checked. If some of them are still available, SI is conducted in order to reduce the number of free time blocks. SI includes two procedures: backward and forward movement to decrease free time blocks and reduction in the amount of resources for malleable tasks.

The two moving procedures are carried out sequentially. Let the current schedule be π. First, all tasks are moved backwards according to the non-increasing order of the finish times of π. The tasks are kept on the priority list $\mathcal{L}_B$. The head task $\mathcal{L}_B^{[1]}$ (the current task with the largest finish time), s_i and f_i are calculated by Equations (6) and (7)). $\mathcal{L}_B^{[1]}$ is

denoted as $v_{[1]}$ and removed from $\mathcal{L}_B$. All successors of $v_{[1]}$ have been calculated before $v_{[1]}$ 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 malleable task, the amount of its on-demand resources is also decreased one by one. The corresponding resource rental costs are calculated for all possible schedules. The start time and the resource amount with the minimum costs of $v_{[1]}$ are updated. $v_{[1]}$ is removed from $\mathcal{L}_B$. The procedure is repeated until $\mathcal{L}_B$ is empty. Then all tasks are moved forward according to the non-decreasing order of start times in the current schedule. The decreasing strategy of start times is similar to the increasing one in the backward movement process. The new start time t is increased one by one from $s_{[1]}$ to $est_{[1]}$. Details of the SI procurement are described in Algorithm 4 of which the time complexity is $O(n^4)$.

D. Multiple sequence generation (MSG)

Generally, the task allocation sequence greatly influences the performance of SC. SC adopts upward-rank values to obtain the task allocation sequence, which does not always result in the best possible schedule. To obtain more task allocation sequences, the variable neighborhood search (VNS) method is proposed. We use the insertion operator to change the sequences, e.g., the operator $I(a,b),\ (1<a<n, 1<b<n, a\neq b, b-1)$ removes the task in position a and inserts it into position b in the sequence. Let the original task allocation sequence be $\bar{s}$ and the new one be $\bar{s'}$. The r-insertion neighborhood $N_r(\bar{s})$ is a set in which each element means performing the insertion operator r times on $\bar{s}$. In this paper, the maximum number of insertions are set to K, i.e., $1 \leq r \leq K$. This implies neighborhoods of increasing size and therefore the variable neighborhood search schema.

Based on the initial sequence s obtained by SC, λ new sequences are generated for each neighborhood $N_r(\bar{s})$. Once the new sequence is obtained, SC is invoked again to construct a new schedule. The cost of the new schedule is calculated and compared with the previous one. The new schedule is kept if the cost improves. After the K-insertion, if no better schedule can be found, the algorithm stops. If an improvement is found, the search starts from the first neighborhood. Algorithm 5 shows the details of the VNS process. The time complexity is $O(K\lambda n^3)$ to obtain a new task allocation sequence.

E. Schedule reconstruction (SR)

After VNS, the schedule is reconstructed by schedule reconstruction (SR). SR destroys the schedule by allocating some tasks to cheaper reserved resources. Tasks with on-demand resources are selected and reallocated with probability ω to the reserved ones. The current schedule is replaced by the new one if a lower cost is obtained and the SR procedure stops. The SR procedure gets a higher probability to obtain better solutions by increasing the diversification of the search process. Details about the SR algorithm are shown in Algorithm 6. The time complexity is $O(Hn^3)$ to reconstruct the current schedule.

Algorithm 4: Schedule improvement (SI)

```
1  begin
2      L_B ← Sort tasks in π by non-increasing order of finish times, π^c ← π;
3      repeat
4          v_[1] ← L_B^[1], π' ← π, s'_[1] ← lst_[1], t' ← s_[1];
5          repeat
6              if v_[1] ∈ M then
7                  h ← on-demand resources of v_[1];
8                  repeat
9                      Calculate est_n;
10                     if est_n > D then
11                         break;
12                     h ← h − 1;
13                 until h = 0;
14             π' ← Call SC;
15             Calculate the cost C(π') using Formula 1;
16             if C(π') ≤ C(π) then
17                 C(π) ← C(π'), t' ← s'_[1];
18             s'_[1] ← s'_[1] − 1;
19         until s'_[1] < s_[1];
20         s_[1] ← t', Remove L_B^[1] from L_B;
21     until Lenth(L_B) = 0;
22     if C(π^c) < C(π) then
23         C(π^c) ← C(π), π^c ← π;
24     else
25         Go to step 50;
26     L_F ← Sort tasks by non-decreasing order of start times;
27     repeat
28         v_[1] ← L_F^[1], π' ← π, s'_[1] ← est_[1], t' ← s_[1];
29         repeat
30             if v_[1] ∈ M then
31                 h ← on-demand resources of v_[1];
32                 repeat
33                     Calculate est_n;
34                     if est_n > D then
35                         break;
36                     h ← h − 1;
37                 until h = 0;
38             π' ← Call SC;
39             Calculate the cost C(π') using Formula 1;
40             if C(π') ≤ C(π) then
41                 C(π) ← C(π'), t' ← s'_[1];
42             s'_[1] ← s'_[1] + 1;
43         until s'_[1] > s_[1];
44         s_[1] ← t', Remove L_F^[1] from L_F;
45     until Lenth(L_F) = 0;
46     if C(π^c) < C(π) then
47         C(π^c) ← C(π), π^c ← π, Go to Step 2;
48     if C(π^c) < C(π^best) then
49         π^best ← π^c, C(π^best) ← C(π^c);
50     return π^best;
```

Algorithm 5: Multiple sequences generation (MSG)

Input: task sequence $\bar{s}$.
Output: task sequence $\bar{s}^{'}$.
begin
  $r \leftarrow 0$;
  repeat
    $r \leftarrow r + 1$;
    for $(i = 0; i < \lambda; i \leftarrow i + 1)$ **do**
      Randomly select $\bar{s}^{'}$ from $N_r(\bar{s})$;
      $\pi^{'} \leftarrow$ Call schedule construction method (SC);
      Calculate the cost $C(\pi^{'})$ using Formula 1;
      if $(C(\bar{s}^{'}) < C(\bar{s}))$ **then**
        $\bar{s} \leftarrow \bar{s}^{'}$;
        $r \leftarrow 0$;
        break;
  until $r \leqslant K$;
  return $\bar{s}^{'}$;

Algorithm 6: Schedule resconstruction (SR)

Input: the current schedule π, the reconstruction probability ω.
Output: the new schedule π.
begin
  for $(h = 1; h \leq H; h \leftarrow h + 1)$ **do**
    if $(y_h = 0)$ **then**
      Randomly generate a number $\omega^{'}, \omega^{'} \in [0, 1]$;
      if $(\omega^{'} \leq \omega)$ **then**
        $y_h = 1$;
        $\pi^{'} \leftarrow$ Call schedule construction method (SC);
        Calculate the cost $C(\pi^{'})$ using Formula 1;
        if $(C(\pi^{'}) < C(\pi))$ **then**
          $\pi \leftarrow \pi^{'}$;
          break;
  return π;

F. Illustration example

Figure 5 is taken as an example to illustrate the process of the proposed MEFT. The first step is to determine the amount of reserved resources. According to Equations (13) and (14), $H^r_{\min} = \lceil 14/9 \rceil = 2$ and $H^r_{\max} = \lceil 14/9/0.8 \rceil = 2$. MEFT starts with the lower bound ($H^r = 2$). The priorities of the tasks are calculated using the TASI procedure and the result is (9, 7, 8, 6, 4, 2, 2) which results in the task allocation sequence (1, 3, 2, 4, 5, 6, 7) for this example. v_1 is allocated first. Since there is only one whole free time block, v_1 is allocated to virtual machine 1 during time periods $1, 2, 3$. v_3 is allocated next. The earliest free time block is time periods $1, 2, 3$ on virtual machine 2. v_3 and the next task v_2 are allocated to time periods $1, 2$ and time period 3 on virtual machine 2 respectively. Since v_4 is a malleable task, it can be allocated to multiple virtual machines at the same time. The time slot 4 on virtual machines 1 and 2 is used for the execution of v_4. Task v_5, v_6, v_7 are allocated in the same way. Figure 5 shows the details of the allocation process. The SR operation is not invoked since no on-demand resources are used.

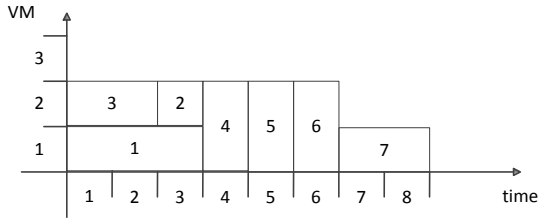


Fig. 5. Example for task scheduling using MEFT.

After several iterations of VNS, a new sequence (1, 3, 2, 4, 5, 7, 6) is found. Tasks are reallocated according to this sequence. The result is shown in Figure 6. While allocating v_6, time slots 6 and 7 on virtual machine 2 are free which forms a new free time block.

v_6 is a rigid task requiring 2 virtual machines and the free time block meets the demands of v_6. If the free time block is not used, $V = 2$ workload is wasted. The workload of v_6 is also $w_i = 2$. Since $V * discount > w_i - V$, the on-demand resource (new virtual machine 3) is rented for v_6. v_6 is processed on virtual machines 2 and 3 at the same time.

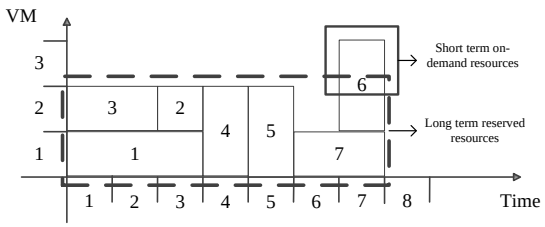


Fig. 6. The schedule of the example by using local search in MEFT.

V. Experimental results

The considered hybrid resource provisioning problem with malleable and rigid tasks has not been studied yet, to the best of our knowledge. The best algorithm BTS for the similar reserved resource provisioning problem [31] is adapted and compared. BTS was proposed to estimate the minimum number of computing hosts. However, BTS only considers the number of virtual machines for each task without considering the allocation of tasks to specific virtual machines. BTS is adapted to BTS-IRPD by adding the Incremental Renting Plan Decision (IRPD) procedure [32] to the schedule of BTS. The best algorithm CPIS-LHCM for the similar on-demand resource provisioning problem [9] is also considered. Two heuristics: Critical Path based Iterative heuristic with Shortest services (CPIS) and List based Heuristic considering Cost minimization and Match degree maximization (LHCM) are developed for the two sub-problems: task-mode (service) mapping and task tabling on renting instances. The LHCM procedure is designed for on-demand resource provisioning

only, so we adapt CPIS to ACPIS to make it suitable for hybrid resource provisioning. Two new aspects are added to CPIS. In order to determine the specific virtual machines for a task, the task with the longest processing time is allocated to virtual machines with the smallest index. The utilization of a virtual machine is calculated to determine the renting alternative of this virtual machine. If it is less than $discount$, the on-demand strategy is used to rent this virtual machine. Otherwise, the reserved alternative is used. In total, four algorithms BTS, BTS-IRPD, CPIS-LHCM and ACPIS are compared to the proposed MEFT. 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.

The Relative Percentage Deviation (RPD) is calculated 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 obtained by the compared algorithms, the RPD of the current algorithm for instance i is calculated as follows:

$$RPD_i = \frac{C(\pi_i) - C(\pi_i^*)}{C(\pi_i^*)} \times 100\% \tag{15}$$

A. Parameter calibration

There are six parameters in the proposed MEFT which need calibration. In the resource initialization (RSI) procedure, two different alternative methods are used to calculate the maximum amount of resources: not considering discount (without) or considering discount (with). In the task allocation sequence initialization (TASI) procedure, four task allocation rules are compared: minimal processing time (1), maximum number of successors (2), minimal slack time (3) and maximum upward-rank value (4). In the schedule improvement (SR) phrase, two alternative methods are considered: not considering schedule procedure (without) and considering schedule improvement (with). In the multiple sequence generation (MSG) procedure, the maximum number of neighborhoods $K \in$\{3, 6, 9, 12, 15\} and the number of solutions $\lambda \in \{6, 10, 14, 18, 22\}$ are calibrated. Finally, we calibrate the reconstruction probability $\omega \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}$ in the schedule reconstruction SC phrase.

We use Rangen [33],[34] to obtain different random workflow instances. The number of tasks n takes values from $\{10, 20, 50, 100, 200\}$. For each value n, 20 instances are generated. The network complexity of the workflow is set as 1.8 according to [34]. The processing time of each task takes a value randomly from a uniform distribution $U(1, 100)$ as the on-demand virtual machines are usually charged for by the hour. 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. The discount of the reserved instance is set as 0.3 according to the ordinary discount of a virtual machine on Amazon EC2 [10]. The deadline for each workflow is supposed to be $D = Est_n \times \theta$, in which θ is a deadline factor and takes a value randomly from $\{1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0\}$.

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.

Figure 7 shows the means plot with 95% Tukey HSD confidence intervals for the resource initialization method, the improvement method and the task allocation rules. The RPD of the method with $discount$ is less than that of the non-$discount$ case. Moreover, the difference is statistically significant as the intervals do not overlap. We use the Reserved Resource Initialization procedure with $discount$ in the following algorithm comparisons. We observe that the RPD of MEFT with improvement is statistically better than that of MEFT with non-improvement. The average difference is a significant 15%. Therefore, we use the improvement method in the following algorithm comparisons. The average RPD of the fourth rule maximum upward-rank value is about 2%. It is better than the minimal slack time rule, the minimal processing time rule and the maximum number of successors rule. We use the maximum upward-rank value rule to obtain the initial task allocation sequence in the following algorithm comparisons.

The plots in figure 8 show the means plots with 95% Tukey HSD confidence intervals for the three parameters K, λ and ω in MEFT. It can be observed that differences in RPDs are statistically significant when $K < 12$ and $\lambda < 14$ whereas RPD differences are not statistically significant when $K \geq 9$ or $\lambda \geq 14$. However, the RPD of MEFT decreases with an increase of K and λ because more solutions are searched for. Additionally, more CPU time is required for large values of these factors. To reach a balance between the CPU time and the RPD value, we set $K = 12$ and $\lambda = 14$ in the following comparisons. When ω takes value 0, there is no reconstruction and MEFT does not perform well. The remaining cases are statistically and significantly different from 0. In addition, differences are not statistically significant among the non-zero ω cases. When $\omega = 0.6$, MEFT obtains the best RPD. Therefore, we use $\omega = 0.6$ in the following algorithm comparisons.

B. Comparison with existing methods

To simulate resource scheduling problems in real clouds, The CloudSim toolkit [35] is used. The toolkit is extended to support on-demand and reserved resources. The VM instances (m4.large) from Amazon EC2 are modeled. 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 as 500 Mbps. The price of on-demand and reserved virtual instances are set according to

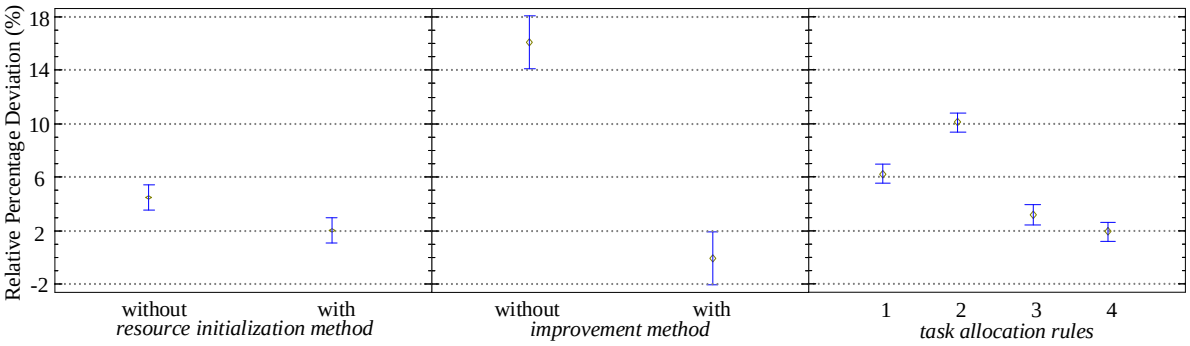


Fig. 7. Means plot with 95% Tukey HSD confidence intervals for the resource initialization method, the improvement method and the task allocation rules in the MEFT calibration.

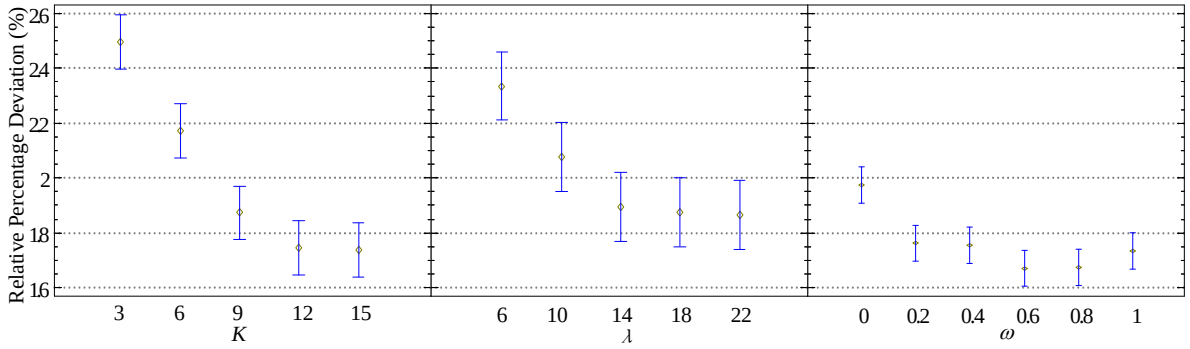


Fig. 8. Means plot with 95% Tukey HSD confidence intervals for the parameters K, λ and ω in the MEFT calibration.

Amazon EC2 [10] with the average discount value 0.3. 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. The virtual machine instances are assumed to be reserved with no upfront cost.

The three scientific workflow instances Montage, LIGO and Epigenomics [1] are adopted to analyze the effectiveness of the proposed MEFT in real environments. Montage has been created by NASA/IPAC 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) is used to generate and analyze gravitational waveforms from data collected during the coalescing of compact binary star systems. The Epigenomics workflow created by the USC Epigenome Center and the Pegasus Team is used to automate various operations in genome sequence processing. Figure 1, Figure 9 and Figure 10 show an example of Montage, LIGO and Epigenomics workflow applications. The nodes mProjectPP, mDiffFit and mBackground in Figure 1 are regarded as the malleable tasks while other nodes are rigid tasks. The nodes TrigBank in Figure 9 are rigid tasks and the other nodes are malleable tasks. The nodes fastQSplit, mapMerge, mapIndex and pileup in Figure 10 are rigid tasks and the other nodes are malleable tasks. The average ratio of rigid tasks in Montage is 30%, while that of LIGO and Epigenomics is 11.1% and 20%, respectively. The instance size of each type of workflow application is set to $n \in \{50, 100, 200, 300, 400\}$. The deadline factors are set in the same way as in the previous sections. For each size and discount value 10 instances are generated. In total, there are $5 \times 10 \times 10 = 500$ instances for performance comparisons.

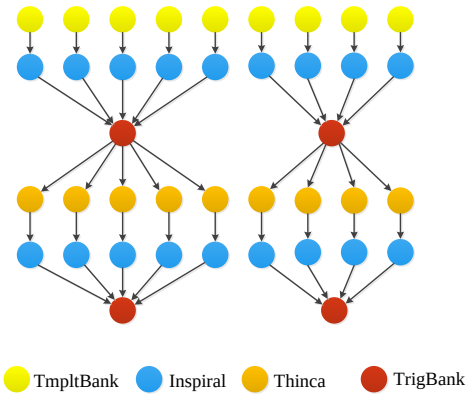


Fig. 9. LIGO workflow application.

For Montage instances, the average CPU times (in milliseconds) of the compared algorithms are shown in Table II. We observe that among all the algorithms ACPIS is the fastest one while MEFT is the slowest algorithm in the comparison. The proposed MEFT takes more time to find suitable new sequences. The other algorithms only find one or several new sequences to obtain a new schedule. Although, MEFT can construct a schedule in $O(n^3)$ time for each sequence, with the increase of the n, the procedure MSG in MEFT takes more time to generate new sequences. However, this paper considers long term workflow scheduling problems and

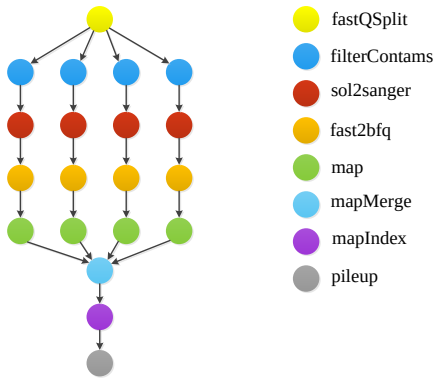


Fig. 10. Epigenomics workflow application.

the workflow applications usually reserve resources for long periods. Compared to the resource rental time, the computing times of the scheduling algorithms are negligible as are about 30 seconds in the worst case for the proposed approach. Therefore, we focus on performance comparisons in this paper.

TABLE II
AVERAGE CPU TIMES (MS.) OF THE COMPARED ALGORITHMS FOR MONTAGE INSTANCES.

n	BTS	BTS-IRPD	CPIS-LHCM	ACPIS	MEFT
50	**147.24**	158.83	318.93	226.37	1106.33
100	450.10	466.34	479.19	**387.86**	3902.74
200	949.73	988.43	697.75	**620.59**	11643.14
300	1298.22	1400.48	823.25	**782.50**	23265.48
400	2075.48	2129.67	1020.03	**982.49**	39164.22
Average	984.15	1028.75	667.83	**599.96**	15816.38

For Montage instances, the interaction plot between n and the compared algorithms is shown in Figure 11. MEFT outperforms the other algorithms in all the instances regardless of n values. Compared to the on-demand only algorithm CPIS-LHCM and the reserved only algorithm BTS, MEFT saves about 40% and 20% in cost respectively. As the value of n increases, the performance of MEFT and other algorithms are almost the same. As n values increase, the cost of the other algorithms increases faster than that of MEFT. This implies that the proposed MEFT is much more suitable for the Montage application with bigger instances.

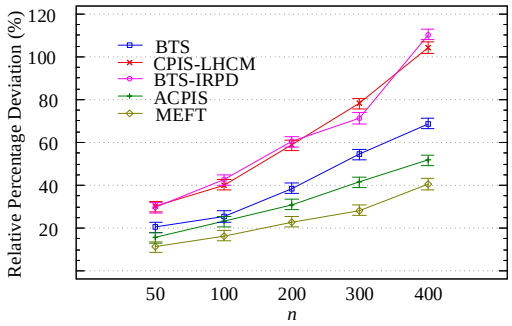


Fig. 11. Interactions between the tested algorithms and the different n values on Montage instances.

For Montage instances, the interaction plot between the deadline factor and the compared algorithms is shown in Figure 12. MEFT outperforms all the other algorithms with different deadline factors. With an increase in the deadline factor, the RPD of all the algorithms does not vary significantly. This implies that the deadline factor has little influence on the performance on the compared algorithms.

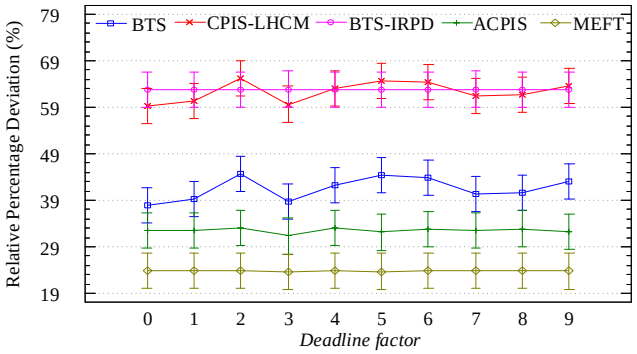


Fig. 12. Interactions between the tested algorithms and the deadline factor on Montage instances.

For LIGO instances, the average CPU times (in milliseconds) of the compared algorithms are shown in Table III. MEFT is the also the slowest one with the average CPU time 10138.71 ms. ACPIS is the fastest one with an average CPU time 384.59 ms. We also just focus on the performance comparisons while the computing times of the scheduling algorithms are negligible compared to the resource rental time.

TABLE III
AVERAGE CPU TIMES (MS.) OF THE COMPARED ALGORITHMS FOR LIGO INSTANCES.

n	BTS	BTS-IRPD	CPIS-LHCM	ACPIS	MEFT
50	**94.38**	101.82	204.44	145.11	709.18
100	288.53	298.93	307.18	**248.63**	2501.76
200	608.80	633.61	447.28	**397.82**	7463.55
300	832.19	897.74	527.73	**501.60**	14913.77
400	1330.43	1365.18	653.87	**629.80**	25105.27
Average	630.87	659.46	428.10	**384.59**	10138.71

For LIGO instances, the corresponding plots are shown in Figure 13. MEFT also outperforms all other algorithms in all instances. On average, MEFT saves about 35% and 25% in costs compared to the on-demand only algorithm CPIS-LHCM and the reserved only algorithm BTS. As n increases, the cost of MEFT increases whereas that of other algorithms has no similar tendency. This implies that the proposed MEFT is much more suitable for the LIGO application with smaller instances.

For LIGO instances, the comparison results with different deadline factors are shown in Figure 14. MEFT outperforms all the other algorithms with different deadline factors. With an increase in the deadline factor, the RPDs of all the algorithms have no monotone increase or decrease trend. Deadline factor

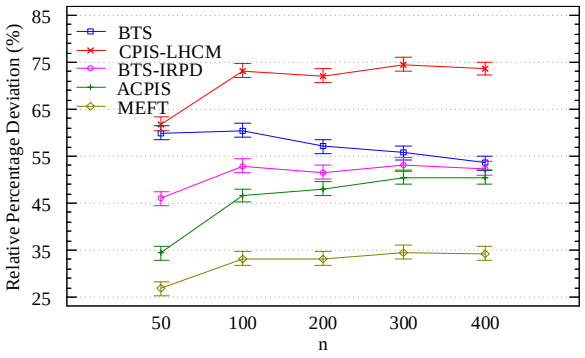


Fig. 13. Interactions between the tested algorithms and the different n values on LIGO instances.

also has little influence on the performance of the compared algorithms.

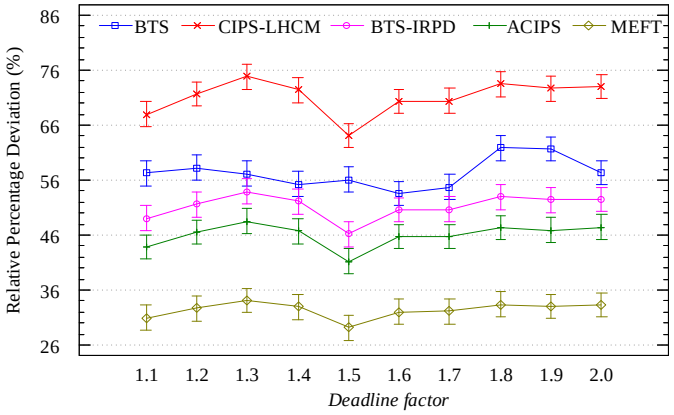


Fig. 14. Interactions between the tested algorithms and the deadline factor on LIGO instances.

For Epigenomics instances, the average CPU times (in milliseconds) of the compared algorithms are shown in Table IV. MEFT is the slower than the competing algorithms with the increase of n. However, since the computing times of the scheduling algorithms are negligible compared to the resource rental time, we also just focus on the performance comparisons.

TABLE IV
AVERAGE CPU TIMES (MS.) OF THE COMPARED ALGORITHMS FOR EPIGENOMICS INSTANCES.

n	BTS	BTS-IRPD	CPIS-LHCM	ACPIS	MEFT
50	**102.04**	110.07	221.02	156.87	766.68
100	311.92	323.17	332.08	**268.78**	2704.60
200	658.16	684.98	483.54	**430.07**	8068.70
300	899.67	970.53	570.51	**542.27**	16122.99
400	1438.31	1475.86	706.88	**680.86**	27140.83
Average	682.02	712.92	462.81	**415.77**	10960.76

For Epigenomics instances, the comparison results with different n values are shown in Figure 15. MEFT is better than all the other compared algorithms for all the instances. On average, the RPD of MEFT is about 50% and 40% less than the on-demand only algorithm CPIS-LHCM and the reserved only algorithm BTS. As n increases, the RPD of different algorithms has no clear tendency. This implies that different n values have little influence on the performance of the compared algorithms.

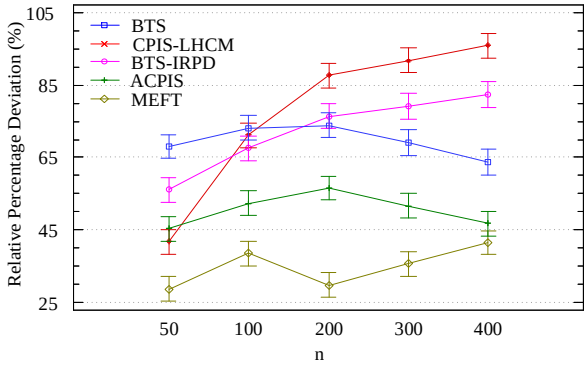


Fig. 15. Interactions between the tested algorithms and the different n values on Epigenomics instances.

For Epigenomics instances, the comparison results with different deadline factors are shown in Figure 16. MEFT outperforms all the other algorithms with different deadline factors. With an increase in the deadline factor, the RPDs of MEFT decreases with some fluctuations while other algorithms have no monotone increase or decrease trend. This implies that MEFT is more suitable for Epigenomics instances with a large deadline factor.

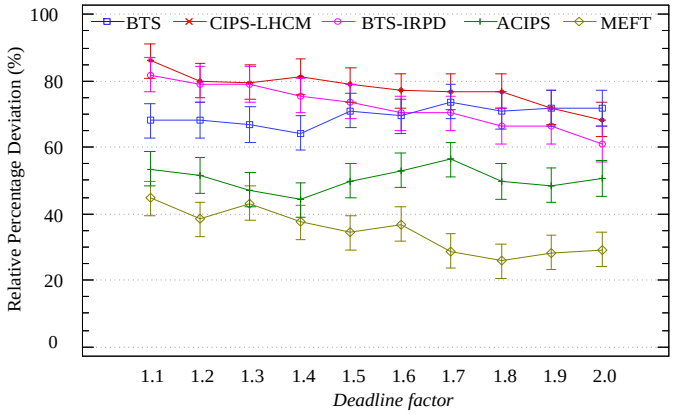


Fig. 16. Interactions between the tested algorithms and the deadline factor on Epigenomics instances.

However, there are some limitations of the proposed MEFT. The CPU time of MEFT grows with the instance size n. For very large values of n, the execution time of MEFT is close to the execution time of the workflow application. In these cases MEFT would not be recommended. For example, for Montage instances, the average execution time of a task is about 10s, the schedule time of a montage instance is about $(n/50)^2$. If $n > 25000$, the scheduling time will be bigger than the execution time.

The above comparisons of Montage, LIGO and Epigenomics instances all show that the proposed MEFT algorithm takes much more CPU time but at the same time it obtains

much better results than other compared algorithms. MEFT takes less CPU time to schedule LIGO than Epigenomics and Montage instances. The reason lies in that there are less rigid tasks in the LIGO instances, the network complexity of LIGO instances is smaller than that of Epigenomics and Montage instances. For the three types of workflow applications with the same instance size, MEFT performs better on LIGO and Epigenomics instances than on the Montage instances. The reason lies in than the schedule improvement (SI) procedure reducing the amount of on-demand instances for malleable tasks and the ratio of malleable tasks in LIGO and Epigenomics instances is larger than that of the Montage instances. The deadline factor has little influence on the performance of the compared algorithms. The reason lies in that as all the algorithms construct a schedule based on the allocation sequences, and the deadline factor influences the performance of different algorithms similarly. Therefore, the RPD value with different deadline factor does not vary significantly.

VI. Conclusions and Future Work

In this paper, a more realistic workflow scheduling problem with both reserved and on-demand instances is considered. A mathematical model with rigid and malleable tasks was established according to the two resource rental strategies. A new multiple sequence-based earliest finish time algorithm is proposed and compared with other adapted state-of-the-art methods from the literature. The experimental and statistical analyses reveal that MEFT is much more suitable for Montage applications with bigger sizes and saves about 40% and 20% in costs compared to the on-demand only and reserved only algorithms. MEFT is also much more suitable for LIGO applications with smaller sizes and saves about 35% and 25% in costs compared to the on-demand only and reserved only algorithms.

For future research, workflow scheduling considering multiple types of virtual machines with hybrid provisioning strategies is a promising topic. More realistic scenarios including multi-clouds with different data transfer times are also worth considering.

Acknowledgment

This work is supported by the National Key Research and Development Program of China (No. 2017YFB1400801), the National Natural Science Foundation of China (Nos. 61572127, 61872077, 61832004) and Collaborative Innovation Center of Wireless Communications Technology. Rubén Ruiz is supported by the Spanish Ministry of Economy and Competitiveness, under the project "SCHEYARD-Optimization of scheduling problems in container yards" (No. DPI2015-65895-R) partly financed with FEDER funds.

References

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

[2] 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*, vol. 13, no. 3, pp. 219–237, 2005.

[3] M. Wieczorek, R. Prodan, and T. Fahringer, "Scheduling of scientific workflows in the ASKALON grid environment," *ACM SIGMOD Record*, vol. 34, no. 3, pp. 56–62, 2005.

[4] Q. Chen, L. Wang, and 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.

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

[6] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities," in *the 10th IEEE International Conference on High Performance Computing and Communications (HPCC'08)*. IEEE, 2008, pp. 5–13.

[7] 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.

[8] 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.

[9] Z. Cai, X. Li, and J. N. Gupta, "Heuristics for provisioning services to workflows in xaas clouds," *IEEE Transactions on Services Computing*, vol. 9, no. 2, pp. 250–263, 2016.

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

[11] 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.

[12] L. Chen, X. Li, and R. Ruiz, "Resource renting for periodical cloud workflow applications," *IEEE Transactions on Services Computing*, 2017.

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] Z. Cai, X. Li, and J. N. D. Gupta, "Critical path-based iterative heuristic for workflow scheduling in utility and cloud computing," in *International Conference on Service-Oriented Computing (ICSOC 2013)*. Springer, 2013, pp. 207–221.

[26] R. Huang, H. Casanova, and A. A. Chien, "Automatic resource specification generation for resource selection," in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*. ACM, 2007, p. 11.

[27] 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.

[28] M. Mazzucco and M. Dumas, "Reserved or on-demand instances? a revenue maximization model for cloud providers," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE, 2011, pp. 428–435.

[29] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, "IaaS reserved contract procurement optimisation with load prediction," *Future Generation Computer Systems*, vol. 53, pp. 13–24, 2015.

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

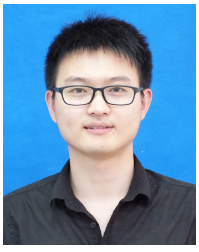
[31] E.-K. Byun, Y.-S. Kee, J.-S. Kim, E. Deelman, and S. Maeng, "Bts: Resource capacity estimate for time-targeted science workflows," *Journal of Parallel and Distributed Computing*, vol. 71, no. 6, pp. 848–862, 2011.

[32] L. Chen and X. Li, "Cloud workflow scheduling with hybrid resource provisioning," *The Journal of Supercomputing*, vol. doi:10.1007/s11227-017-2043-5, pp. 1–25, 2017.

[33] R. Kolisch, A. Sprecher, and A. Drexl, "Characterization and generation of a general class of resource-constrained project scheduling problems," *Management Science*, vol. 41, no. 10, pp. 1693–1703, 1995.

[34] R. Kolisch and A. Sprecher, "Psplib-a project scheduling problem library: Or software-orsep operations research software exchange program," *European Journal of Operational Research*, vol. 96, no. 1, pp. 205–216, 1997.

[35] 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.



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 in 1993 and 1999 respectively, and the Ph.D. degree in Applied Computer Science from the Harbin Institute of Technology in 2002. He is a full professor at the School of Computer Science and Engineering, Southeast University, Nanjing, China. 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 Parallel and Distributed Systems*; *IEEE Transactions on Services Computing*; *IEEE Transactions on Cybernetics*; *IEEE Transactions on Automation Science and Engineering*; *IEEE Transactions on Cloud Computing*; *IEEE Transactions on Systems, Man and Cybernetics: Systems*; *Information Sciences*; *Omega*, *European Journal of Operational Research*; *International Journal of Production Research*; *Expert Systems with Applications* and *Journal of Network and Computer Applications*. His research interests include Scheduling in Cloud Computing, Scheduling in Cloud Manufacturing, Service Computing, Big Data and Machine Learning.

Yucheng Guo received his B.Sc. degree in Computer Science and Technology from Nangjing University of Posts and Telecommunications, Nanjing, China in 2012. He obtained his M.Sc. degree in Computer Science and Engineering from Southeast University in 2015. His research interest is cloud computing.

Rubén Ruiz is a 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.