This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

Dynamic Cloud Task Scheduling Based on a Two-Stage Strategy

PeiYun Zhang, *Member, IEEE*, and MengChu Zhou, *Fellow, IEEE*

***Abstract*— To maximize task scheduling performance and minimize nonreasonable task allocation in clouds, this paper proposes a method based on a two-stage strategy. At the first stage, a job classifier motivated by a Bayes classifier's design principle is utilized to classify tasks based on historical scheduling data. A certain number of virtual machines (VMs) of different types are accordingly created. This can save time of creating VMs during task scheduling. At the second stage, tasks are matched with concrete VMs dynamically. Dynamic task scheduling algorithms are accordingly proposed. Experimental results show that they can effectively improve the cloud's scheduling performance and achieve the load balancing of cloud resources in comparison with existing methods.**

***Note to Practitioners*— Task scheduling is one of the challenging problems in cloud computing, especially when deadline and cost are considered. As an important actuator, virtual machines (VMs) play a vital role for cloud task scheduling. To meet task deadlines, one needs to save the time of creating VMs, task waiting time, and executing time. To minimize the task execution cost, one needs to schedule tasks onto their most suitable VMs for execution. We propose a cloud task scheduling framework based on a two-stage strategy to do so. It precreates VMs according to historical scheduling data, therefore saving time for tasks to wait for creating VMs. It matches tasks with their most suitable VMs dynamically, therefore saving their execution cost. Under the premise of meeting task deadlines, it minimizes the waiting time of VMs to schedule tasks, thus minimizing the cost to be paid by users who utilize VMs. The readily deployable algorithms are designed and illustrated to improve cloud task scheduling and execution results in comparison with those using traditional methods.**

***Index Terms*— Clouds, dynamic scheduling, task classifier, task scheduling, virtual machines (VMs).**

I. Introduction

CLOUD computing, called clouds for short, has grown exponentially in the business and research community for a number of years. It has become popular due to recent advances in virtualization technologies [1]. Clouds provide services to users who do not own sufficient computing resources. They achieve the economy of scale via multiplexing. Because clouds provide a finite pool of virtualized on-demand resources, optimally scheduling them has become an essential and rewarding research topic [2]. In clouds, various applications are submitted to data centers to obtain some services on a pay-per-use basis. However, scheduling workload in heterogeneous cloud environment is very challenging due to limited cloud resources with varying capacities and functionalities [1]. The key issue is how to allocate user tasks to maximize the profit of cloud service providers while guaranteeing quality-of-service (QoS) for all the tasks [3]. In a dynamic environment, how to successfully schedule tasks and make high utilization of cloud resources is an important problem.

Task scheduling is to implement the reasonable deployment of cloud resources to meet user requirements while maximizing the economic benefits of cloud service providers. As a service business model, task scheduling strategies in clouds are needed to meet the QoS requirements of user tasks [4]–[16], which include deadline [3], [7], [11], makespan [1], [12], and cost. At the same time, service profit and energy cost for cloud service providers should be fully considered [3], [13]–[16]. Improperly matching applications at some hardware platforms can degrade the overall performance of clouds and may violate the QoS guarantees that many user tasks require [9].

Exhaustive-search-based scheduling for clouds is impossible. Its complexity grows exponentially with the numbers of tasks and resources. Some scholars use intelligent optimization algorithms to find an approximate optimal solution. Such algorithms [2], [8] can reduce their search space and ensure their execution within a feasible operational time, which provides a compromise between their running time and resultant schedule's optimality. Particle swarm optimization [3], [38], [39], genetic algorithms [17], ant colony [18] and Cuckoo-search [40] are their representative examples.

There are several classical scheduling algorithms, such as Min–Min [19]–[21], Max–Min [22], and first-in–first-out (FIFO) [23]. The first two fail to utilize resources efficiently, thereby leading to a load imbalance problem. They also fail to cope with user-priority during scheduling [24]. FIFO [23] schedules tasks by the order of task submissions. If a submitted task needs to occupy a large number of computing resources and time span is large, the tasks with small time span must wait for a long time. It thus affects user experience negatively.

Clouds have several characteristics such as dynamicity and large scale. For cloud service providers, physical hosts may be

Manuscript received February 26, 2017; accepted April 4, 2017. This paper was recommended for publication by Associate Editor D. Down and Editor S. Reveliotis upon evaluation of the reviewers' comments. This work was in part supported by the National Natural Science Foundation of China under Grants 61472005 and 61201252 and FDCT (Fundo para o Desenvolvimento das Ciencias eda Tecnologia) under Grant 119/2014/A3. *(Corresponding author: MengChu Zhou.)*

P. Y. Zhang is with the School of Mathematics and Computer Science, Anhui Normal University, Wuhu 241003, China (e-mail: zpyanu@ahnu.edu.cn).

M. C. Zhou is with the Institute of Systems Engineering, Macau University of Science and Technology, Macau 999078, China and also with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102 USA (e-mail: zhou@njit.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASE.2017.2693688

1545-5955 © 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

permitted to come in or move away from clouds dynamically; for cloud service users, their tasks are permitted to submit and get scheduled dynamically. In general, task distribution is not uniform, thereby raising needs to manage and schedule tasks well in clouds. Tasks reaching a cloud system are often diverse and heterogeneous with varying QoS requirements [24]. As clouds have been widely deployed in various fields, their reliability and availability become the major concerns of cloud service providers as well as users [25], [26]. Zhu *et al.* [27] devise an agent-based scheduling mechanism in cloud computing environment to allocate real-time tasks and dynamically provide resources. He *et al.* [28] propose a method that first assigns high-priority jobs to their preferred cloud resources, and then low-priority jobs to the remaining ones. These two methods need to decrease cost due to mismatched tasks and resources.

In clouds, virtual machines (VMs) are very important service resources for task scheduling. During scheduling, we face two kinds of problems. The first one is when a relatively larger task is assigned to a VM with weak processing ability, its processing time is relatively longer and, sometimes, the task may not be completed before its deadline. This may interrupt the whole task sequence. The second one is when a small task is assigned to a VM with strong processing ability; it may make a large task stay in a waiting state for an extra time. Therefore, thereby reducing the overall throughput of clouds.

Ideally, tasks should be scheduled at their most suitable VMs. One way to do so is to create proper VMs temporarily according to the exact requirements of tasks. Unfortunately, it takes too much time to do so during scheduling. If concrete VMs are dynamically created, the time cost can be significant, sometimes up to half an hour. Therefore, it is too long for users to wait for scheduling their tasks. Thus, one motivation of our work is to decrease such time cost by creating VMs beforehand based on historical task data. We also propose new algorithms to improve the performance of task scheduling.

The major contributions of this paper are as follows. To resolve the mentioned issues, this paper presents a two-stage strategy to create VMs beforehand based on historical scheduling data and takes task requirement (such as deadline and cost) into scheduling consideration. It gives a task scheduling method to minimize the completion time of tasks subject to deadline constraints while maintaining the highest level workload balance among all the cloud resources in a dynamic cloud environment. We have compared the proposed method with its peers through computer simulations.

Section II presents a scheduling framework and a two-stage task scheduling strategy. Section III discusses the implementation of the proposed framework and scheduling algorithms. Section IV gives experimental results and makes the comparisons with some existing methods. Finally, Section V draws the conclusions and indicates work.

II. Scheduling Framework

Let V be a VM set, $V = \{1, 2, \ldots, N\}$ and $v_i, i \in \{1, 2, \ldots, N\}$, represent the ith VM. v_i has four attributes denoted as v_i^α, $\alpha \in \{1, 2, 3, 4\}$. They represent CPU resources (such as clock speed of CPU), memory resources, network bandwidth, and hard disk storage, respectively. Hence, we have $v_i =< v_i^1, v_i^2, v_i^3, v_i^4 >$.

Let $T = \{1, 2, \ldots, M\}$ be a task set provided by users and $t_j, j \in \{1, 2, \ldots, M\}$, represent the jth task. $t_j =< t_j^{id}, t_j^r, t_j^f, x_j >$, where

1) t_j^{id} is the unique ID of task j.
2) t_j^r represents requirements of task j. $t_j^r =< t_j^1, t_j^2, t_j^3, t_j^4 >$ specifies t_j's requirements for CPU, memory, network bandwidth and hard disk storage.
3) t_j^f indicates t_j's deadline. When t_j's deadline is violated, task scheduling fails.
4) x_j represents the priority of task j. If t_j is a rush or high-paying user's job, it is of high priority and $x_j = 1$; otherwise, if it is a regular job, $x_j = 0$.

In this paper, the clouds consist of a set of hosts that is labeled as H. Let $H_k, k \in \{1, 2, \ldots, K\}$, represent the kth host that can create G_k VMs, i.e., $H_k = \{v_{nk} | n \in \{1, 2, \ldots, G_k\}\}$. v_{nk} represents the nth VM of the kth host. VMs are generated from H_k by virtualization.

Task scheduling is defined as a function mapping tasks to VMs

$$f : T \to V$$

After successful scheduling, each task is scheduled and executed at a suitable VM. To decrease the mapping complexity, we divide tasks to fit to VM types. Let $U = \{1, 2, \ldots, L\}$ be a set of VM types. L denotes the number of VM types. Each successfully created VM belongs to a VM type.

Next, we utilize a task classifier to realize the mapping from tasks to VM types. This way can reduce the difficulties of task scheduling as the number of VM types can be much smaller than that of VMs. We define a binary variable

$$y_{jik} = \begin{cases} 1, & t_j \text{ is assigned to } v_{ik} \\ 0, & \text{otherwise.} \end{cases}$$

Task guarantee ratio at hosts is given as follows:

$$\phi_H = \sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{jik} / M.$$

Given host H_k, task guarantee ratio at its VMs is as follows:

$$\phi_K = \sum_{i=1}^{N} \sum_{j=1}^{M} y_{jik} / M.$$

We may define priority tasks (serving VIP users and urgent/paying-high-price users) and ordinary tasks (for ordinary users and noncritical jobs). Priority task guarantee ratio at hosts is given as follows:

$$\psi_H = \sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{jik} \cdot x_j / \sum_{j=1}^{M} x_j.$$

Given host H_k, priority task guarantee ratio at VMs is

$$\psi_K = \sum_{i=1}^{N} \sum_{j=1}^{M} y_{jik} \cdot x_j / \sum_{j=1}^{M} x_j.$$

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

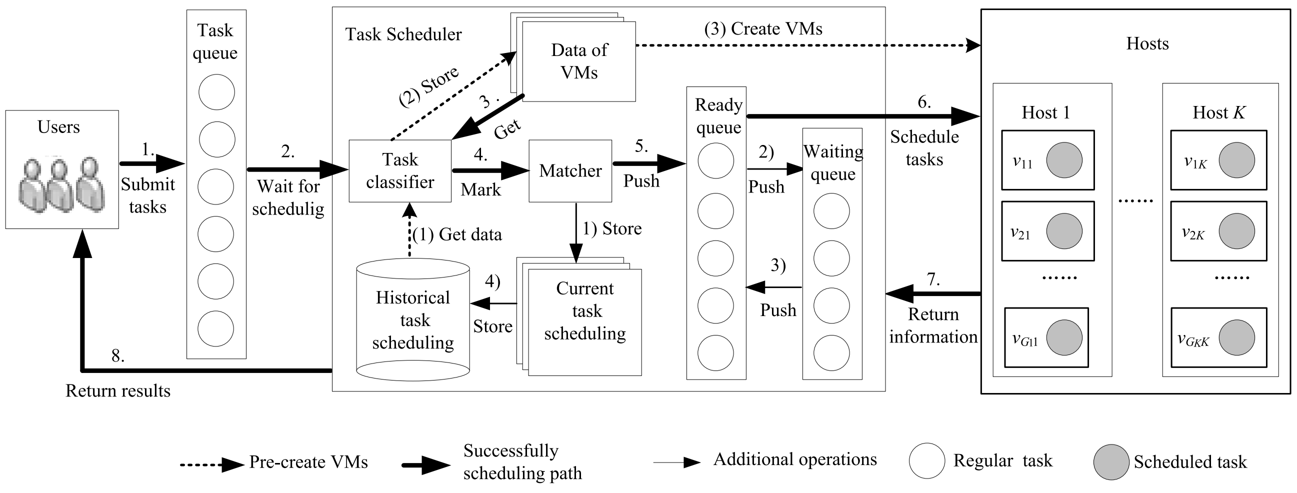


Fig. 1. Cloud task scheduling framework based on a two-stage strategy.

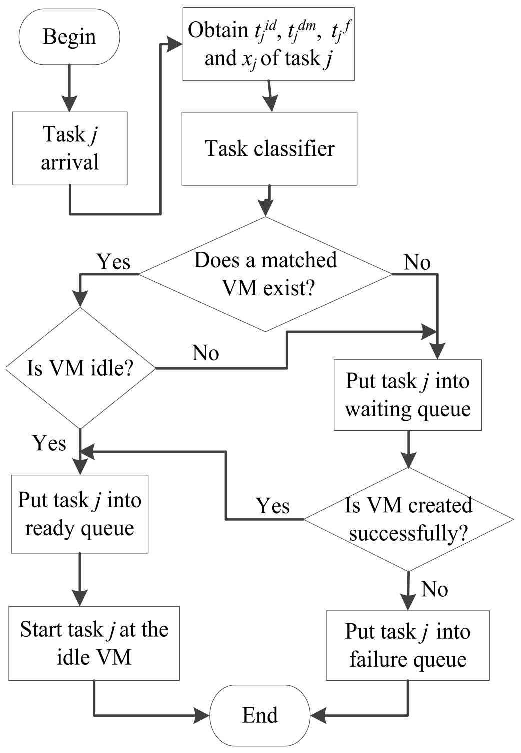

Begin
Task j arrival
Obtain t_j^{id}, t_j^{dm}, t_j^{f} and x_j of task j
Task classifier
Does a matched VM exist?
Yes
No
Is VM idle?
No
Yes
Put task j into waiting queue
Put task j into ready queue
Is VM created successfully?
Yes
No
Start task j at the idle VM
Put task j into failure queue
End

Fig. 2. Scheduling task j with more details.

We define our task scheduling framework for clouds as shown in Fig. 1. It has the following four main modules.

1) *Task Classifier:* The proposed task classifier has two functions in two stages. At the first stage, it determines VM types and their quantities by analyzing historical task scheduling information. At the second stage, it intends to classify a submitted task and match it to the most suitable VM type. Once a task is matched, we call that it is "marked." Its type is defined as its matched VM's type. The detailed algorithms of creating VM types and matching a job with a VM type are to be discussed in the next section (Algorithms 1 and 2).
2) *Matcher:* It matches tasks with suitable concrete VMs based on the results of the task classifier. It is implemented via Algorithm 3.
3) *Ready Queue:* When there are idle VMs meeting the requirements of tasks, tasks are pushed into a ready queue and may be executed immediately. It is materialized via Algorithm 3.
4) *Waiting Queue:* When there are no idle VMs meeting the requirements of tasks, tasks are pushed into a waiting queue and wait for concrete VMs during a scheduling process. Its process is shown in Fig. 2 to be discussed later.

Algorithm 1: Create VM Types and Create VMs at Hosts

```
Input: DB. /*DB is historical scheduling data*/
Output: Types. /* a set of VM types*/
1: Types ← ∅;
2: $\tilde{T}$ ← Data processing of DB;
3: L ← Task types count of $\tilde{T}$;
4: For i =1 to L
5:     compute $P\left(\tilde{T}_i\right)$;
6: End For
7: K ← TopK($P(\tilde{T}_i)$);
8: For i=1 to K
9:     Types← (create VM types based on value of $P(\tilde{T}_i)$);
10:    $v_i$ ←create VM i;
11: End For
12: Output Types;
```

In Fig. 1, a two-stage strategy is applied.

1) *First Stage:* As shown via dotted directed arcs, we have the following steps.
   a) Based on historical task scheduling information, we obtain historical data.
   b) We propose a task classifier to classify tasks and store them in a database (data of VMs in Fig. 1), and create a set of VM types.
   c) We create a proper number of VMs of different types at hosts.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

Algorithm 2: Mark Tasks With VM Types

```
Input: $T$, $Types$. /*$T$ is an initial task set. $Types$ comes from Algorithm 1*/
Output: $T^{\text{flag}}$. /* information of marked tasks*/
1: $T^{\text{flag}} \leftarrow \emptyset$;
2: $K \leftarrow |Types|$;
3: $T \leftarrow$ Rank tasks in $T$ by $x_j$ in descending order;
4: For $j$ =1 to $T.\ sizeof()$
5:     $t_j \leftarrow < t_j^{id}, t_j^r, t_j^f, x_j >$;
6:     $\delta \leftarrow 0$;
7: For $i$ =1 to $K$
8:     $P\left(Y_i|t_j\right) \leftarrow \prod_{\alpha=1}^4 P\left(t_j^\alpha|v_i^\alpha\right)$;
9:     If $((i', j') == arg\ max\{P(Y_i|t_j)\}$ & $t_j$ can be executed before $t_j^f)$
10:        $\delta \leftarrow 1$;
11:        $T_j^{\text{flag}} \leftarrow < Y_{i'}, t_{j'}, \delta >$;
12:    End If
13: End For
14:    If ( $\delta$ ==0)
15:        $Y_x \leftarrow$ (a proper VM type is created);
16:        $T_j^{\text{flag}} \leftarrow < Y_x, t_j, \delta >$;
17:    End If
18: End For
19: Output $T^{\text{flag}}$;
```

Algorithm 3: Match Tasks With VMs

```
Input: $T^{\text{flag}}$. /*$T^{\text{flag}}$ comes from Algorithm 2*/
Output: $TS$. /* matched pairs of tasks and VMs*/
1:  $n \leftarrow T^{\text{flag}}.\ sizeof\ ()$;
2:  For $j = 1\ to\ n$ & each $T_j^{\text{flag}} \in T^{\text{flag}}$
3:      $U \leftarrow T_j^{\text{flag}}.\ U$; $t_j \leftarrow T_j^{\text{flag}}.\ TASK$;
4:      $\delta \leftarrow T_j^{\text{flag}}.\ \delta$;
5:      If (there is a free VM $i$ of type $Y_i$)
6:          $status \leftarrow \mathbf{1}$;
7:      End If
8:  If (sequence of ready is not full)
9:      $ready \leftarrow \mathbf{1}$;
10: End If
11: If ($status$==1 & $\delta$ ==1 & $ready$==1)
12:     $sequence\text{-}of\text{-}ready \leftarrow < v_i, t_j >$;
13: End If
14: If ($status$==0&$\delta$ ==1& $ready$==1)||($\delta$ ==0)
15:     create a new VM $i$ of type $Y_i$;
16:     $v_i \leftarrow < v_i^1, v_i^2, v_i^3, v_i^4 >$;
17:     $sequence\text{-}of\text{-}ready \leftarrow < v_i, t_j >$;
18: End If
19: Update information of task scheduling;
20: $TS \leftarrow < t_j, v_i >$ ;
21: End For
22: Output $TS$;
```

2) *Second Stage:* As shown via thick directed arcs in Fig. 1, we have the following steps.
   a) Users summit tasks to the system dynamically, which are stored into a task queue.
   b) The task classifier fetches rush and then regular tasks from the task queue.
   c) The task classifier retrieves the type information of VMs as created at the first stage. Then it marks each of the tasks with a suitable VM type.
   d) The marked tasks are passed to matcher that matches marked tasks with concrete VMs of proper types.
   e) The matched pair <task, VM> is pushed into a ready queue.
   f) The tasks are scheduled and executed at VMs.
   g) The scheduling information is returned to task scheduler (and stored later as the historical information).
   h) Task execution results are returned to users.

The entire path with thick solid directed arcs shows a successful task scheduling result. We have some additional operations in the proposed framework as follows.

1) Results about successful and unsuccessful task scheduling are stored into a database with historical task information.
2) Tasks in the ready queue are pushed back into a waiting queue when their scheduling conditions are no longer satisfied.
3) When proper VMs are available, tasks in the waiting queue are pushed into the ready queue if the ready queue has spaces.
4) When tasks are scheduled successfully, such information including $< t_j, v_i >$ and execution time is stored into the historical scheduling database.

Our framework has several kinds of queues, such as task, ready, and waiting queues. Its tasks have two priorities only. Given their same priority, tasks are ranked by their deadline.

III. Scheduling Algorithms

We design task scheduling algorithms by following the proposed two-stage strategy.

A. First Stage: Mark Tasks

Let $\tilde{T}$ be a historical task set extracted from the historical task scheduling database DB in a cloud computing system, as shown in Fig. 1.

Let $\tilde{T}_l$ represent a set of type-l tasks. Ratio $P(\tilde{T}_l)$ is given as follows:

$$P(\tilde{T}_l) = \frac{|\tilde{T}_l|}{|\tilde{T}|}. \tag{1}$$

We denote the set of the tasks processed historically by a type-lVM as

$$\Omega_l = \{i : \text{task } i \text{ is a type-}l \text{ task processed by a type-}l \text{ VM}\}.$$

Table I shows some historical examples of task types. There are four task types. For task type 1, $|\tilde{T}_1| = 100$. $|\tilde{T}| = 100 + 200 + 100 + 600 = 1000$. Based on Table I and (1), we have $P(\tilde{T}_1) = 0.1$.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE I
PARAMETERS OF TASKS

Task type	Task requirements	Task deadline	Task count
1	100 floating point operations,1M memory	submission time+100ms	100
2	200 floating point operations, 10M memory	submission time +300ms	200
3	500 floating point operations, 50M memory	submission time +500ms	100
4	1000 floating point operations, 100M memory	submission time +1000ms	600

TABLE II
PARAMETERS OF VMs

VM type	Attributes of VMs				VM count
1	10MHz	10M	48Kbps	10M	5
2	100MHz	100M	96Kbps	100M	10
3	200MHz	200M	128Kbps	200M	10
4	400MHz	400M	256Kbps	400M	5

Accordingly, we can precreate a certain number of VMs with obtained VM types to save task scheduling time, as shown in Algorithm 1.

Given task j and VM i, we have their attributes, i.e., $t_j^r = \{t_j^\alpha | \alpha \in \{1, 2, 3, 4\}\}$, and $v_i = \{v_i^\alpha | \alpha \in \{1, 2, 3, 4\}\}$. The matching degree $P(t_j^\alpha | v_i^\alpha)$ is computed as follows:

$$P\big(t_j^\alpha | v_i^\alpha\big) = \begin{cases} \big(v_i^\alpha / t_j^\alpha\big)^{2,} & \text{if } t_j^\alpha > v_i^\alpha \\ \big(v_{\max}^\alpha - v_i^\alpha + t_j^\alpha\big)/v_{\max}^\alpha, & \text{otherwise} \end{cases} \tag{2}$$

where $v_{\max}^\alpha = \max_{k \in U} v_k^\alpha$ and k represents type k VMs.

In (2), "$\alpha = 1$" represents CPU resources; "$\alpha = 2$" represents memory resources; "$\alpha = 3$" represents network bandwidth; and "$\alpha = 4$" represents hard disk storage. Because $v_i^\alpha > 0$, the matching degree $P(t_j^\alpha | v_i^\alpha) > 0$.

The intention of (2) is that it prefers a task requirement's overmatch over undermatch when a perfect match cannot happen. For example, assume that $v_i^1 = 11$, $v_k^1 = 9$, $v_{\max}^1 = 100$, and $t_j^1 = 10$. According to (2), we can obtain the matching degrees as follows:

$$\begin{aligned} P\big(t_j^1 | v_i^1\big) &= \big(\upsilon_{\max}^1 - v_i^1 + t_j^1\big)/v_{\max}^1 \\ &= (100 - 11 + 10)/100 = 0.98. \\ P\big(t_j^1 | v_k^1\big) &= \big(v_k^1 / t_j^1\big)^2 = (9/10)^2 = 0.81 \\ P\big(t_j^1 | v_{\max}^1\big) &= \big(v_{\max}^1 - v_{\max}^1 + t_j^1\big)/v_{\max}^1 \\ &= (100 - 100 + 10)/100 = 0.1. \end{aligned}$$

We can see that $0.98 > 0.8 > 0.1$. It means that for t_j^1, it has the better matching with v_i^1 (slightly overmatch), but neither v_k^1 (slightly undermatch) nor $v_{\max}^1$ (too much overmatch). Note that if $t_j^\alpha = v_i^\alpha$ or $P(t_j^\alpha | v_i^\alpha) = 1$, we have a perfect match.

Table II shows the example of attributes of VMs and VM types. In it, there are four types of VMs. Each type has a certain number of VMs, e.g., type 1 has five VMs. Their four attributes are given. Detailed relationship between Tables I and II is to be revealed later. Let Y_i represent VM type i.

For task j, motivated by a Bayes classifier, we can compute the possibility of task j belonging to type Y_i, denoted as $P(Y_i | t_j)$, as follows:

$$P(Y_i | t_j) = \prod_{\alpha=1}^{4} P\big(t_j^\alpha | v_i^\alpha\big). \tag{3}$$

In Algorithm 2, the decision function of task j is as follows:

$$(i', j') = \arg\max P(Y_i | t_j). \tag{4}$$

In (4), $t_{j'}$ and $Y_{i'}$ are obtained and they achieve the best matching. Note that our task classifier is designed based on task features/attributes. If task j's requirements meet perfectly with a VM type's attributes, i.e., $P(t_j^\alpha | v_i^\alpha) = 1$, $\forall \alpha \in \{1, 2, 3, 4\}$ or $P(Y_i | t_j) = 1$, then task j will definitely select a type i VM. When it is not matched, for example, one with larger capacity and another with smaller one, our classifier design tends to favor the choice of a VM with larger capacity for a task as mentioned before.

For example, assume that $[t_j^1, t_j{}^2, t_j{}^3, t_j{}^4] = [10, 10, 48, 10]$. Based on Table II, we can see that t_j^α is equal to v_1^α. According to (2), we obtain $P(t_j^\alpha | v_1^\alpha) = 1$ for all $\alpha \in \{1, 2, 3, 4\}$. Then, we have

$$P(Y_1 | t_j) = \prod_{\alpha=1}^{4} P\big(t_j^\alpha | v_1^\alpha\big) = 1.$$

When $i \in \{2, 3, 4\}$, we can see that $t_j^\alpha < v_i^\alpha$. Thus, we compute

$$\begin{aligned} P(Y_2 | t_j) &= \prod_{\alpha=1}^{4} P\big(t_j^\alpha | v_2^\alpha\big) \\ &= P\big(t_j^1 | v_2^1\big) P\big(t_j^2 | v_2^2\big) P\big(t_j^3 | v_2^3\big) P\big(t_j^4 | v_2^4\big) \\ &= \frac{400 - 100 + 100}{400} \cdot \frac{400 - 100 + 10}{400}. \\ &\quad \times \frac{256 - 96 + 48}{256} \cdot \frac{400 - 100 + 10}{400} 0.378. \end{aligned}$$

In the same way, we obtain

$$P(Y_3 | t_j) = \prod_{\alpha=1}^{4} P\big(t_j^\alpha | v_3^\alpha\big) = 0.100$$

$$P(Y_4 | t_j) = \prod_{\alpha=1}^{4} P\big(t_j^\alpha | v_4^\alpha\big) \cong 0.$$

Now we can conclude that $P(Y_1 | t_j)$ reaches the maximum. Hence, task j can be marked as VM type 1.

Based on the above analysis, when an initial set of tasks arrives at task scheduler, the proposed task classifier divides all tasks into proper types. Then, each task is marked with its corresponding VM type as shown in Algorithm 2.

Algorithm 2 is based on the task classification results from Algorithm 1. It mainly consists of two parts. The first part from Lines 7 to 13 decides when task j can be executed before deadline t_j^f. The second one from Lines 14 to 17 handles the opposite case.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE III
STRUCTURE OF T^{flag}

U	TASK	δ
VM types	Tasks of an initial task set	It marks whether there exist concrete VMs

In Line 1, T^{flag}, named marked tasks, consists of three parts, as shown in Table III in which

$$\delta = \begin{cases} 1, & \text{if at least one concrete VM exists} \\ 0, & \text{otherwise.} \end{cases}$$

B. Second Stage: Match Tasks With VMs

The first stage is mainly to mark tasks with suitable VM types. During an actual scheduling process, we need to match them with specific VMs dynamically. This is the work of second stage as shown in Algorithm 3.

In Algorithm 3, when we match tasks with VMs of marked types, specific VMs may be idle or busy. We need to treat such situations differently. Hence, Algorithm 3 has two parts. The first part deals with the one when type-i VMs are available. If ($\delta == 1$ and *ready*==1), and if there is a free concrete type-i VM, matching is successful as shown from Lines 11 to 13.

The second part deals with the situation when a proper-type VM i is occupied by other tasks or it does not exist. If no VM meets requirements of task j, the scheduling process is interrupted. Hence, some rescue measures are needed and new VMs must be created, as shown from Lines 14 to 18.

In Algorithm 3, pair $\langle t_j, v_i \rangle$ represents that task j is matched with VM i. The meaning of symbols "*U*," "*TASK*," and "δ" in Lines 3 and 4 is explained in Table III.

C. Dynamic Task Scheduling

A more detailed process of executing task j is shown in Fig. 2. The proposed task classifier is used. When a user submits task j, task j is classified according to its requirement, and marked with a VM type, when there is no VM or when the matched VM is busy, the waiting queue is used to buffer a job.

When a right-type VM and ready queue spaces are available, tasks in the waiting queue are pushed into the ready queue. In a given time, when the requirements of the tasks in the waiting queue cannot be met, task scheduling should be interrupted and these tasks are pushed into a failure queue. The failure queue is used to store unsuccessfully scheduled tasks. In Fig. 2, the tasks in the ready queue are scheduled before their deadlines are reached.

When scheduling task j at VM i, let $S(t_j, v_i)$ be its latest start time and $E(t_j, v_i)$ be its execution time of task j at v_i

$$S(t_j, v_i) = t_j^f - E(t_j, v_i) \tag{5}$$

where t_j^f is the deadline of task j.

Let $A(t_j, v_i)$ be actual start time of task j at VM i. We first schedule the tasks from the ready queue at the corresponding idle timestamp, such as $S(t_j, v_i)$ of VMs. Hence, we ensure that tasks can be executed before their deadline. Then, if there is much idle time before the timestamp, we schedule tasks before timestamp as early as possible.

Algorithm 4: Dynamic Cloud Task Scheduling Based on a Two-Stage Strategy

```
Input: $TS$./*$TS$ comes from Algorithm 3*/
Output: $scheduled$. /*scheduling info. */
1: $scheduled[] \leftarrow \emptyset$;
2: $n \leftarrow |TS|$;
3: For($j = 1$ to $n$ & each pair $<t_j, v_i> \in TS$ at host $k$)
4:     $y_{jik} \leftarrow 0$;
5:     compute $S(t_j, v_i)$;
6:     compute idle time $slot_j$ of VM $i$ before $S(t_j, v_i)$;
7:     assign task $j$ to VM $i$ during $slot_j$;
8:     $y_{jik} \leftarrow 1$;
9: End For
10: For $j = 1$ to $n$
11:   If $A(t_j, v_i)$ reaches the minimum among all $S(t_k, v_i), k = 1, 2, \ldots, n$,
12:     schedule task $j$ at VM $i$ during $slot_j$;
13:     $scheduled[j] \leftarrow (< t_j, v_i > \& E(t_j, v_i) \& F(t_j, v_i))$;
14:   End If
15: End For
16: Output $scheduled$;
```

Algorithm 4 consists of two parts. The first one from Lines 3 to 9 assigns tasks to idle time slot of VMs. The second one from Lines 10 to 15 schedules tasks at the idle time slot of VMs as early as possible while meeting the constraints. Our purpose is to deal with fault problems that cause task scheduling latency. According to [27], much research has been conducted to realize fault tolerance in distributed systems, among which fault-tolerant scheduling plays a significant role. Therefore, as shown in Lines 5 and 6, we compute $S(t_j, v_i)$ and the idle time $slot_j$ of VM i before $S(t_j, v_i)$. We assign t_j to VM i at $slot_j$ in Line 7.

Let $F(t_j, v_i)$ be t_j's the latest completion time at matched VM *i*. The following cost objective is considered when scheduling tasks:

$$\begin{aligned} &\min \ \text{cost}(t_j) \\ &\text{subject to } t_j^f - F(t_j, \ v_i) \geq 0 \end{aligned} \tag{6}$$

where cost(t_j) represents the execution cost of task j at VM i. Different VMs of different types have different prices. Duan *et al.* [4] give some examples about Google cloud and Amazon EC2 VMs. We can see that VMs are paid in hours. Therefore, in Algorithm 4, we just take the running time as the optimization objective. The time can be converted to cost (charged to users). Consequently, we form the following

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

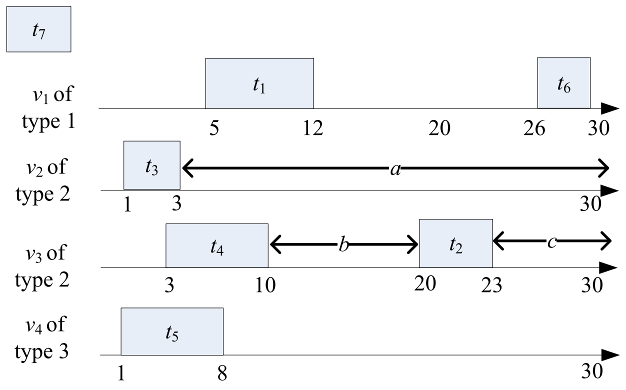


Fig. 3. Before scheduling task t_7.

problem instead:

$$\begin{aligned}&\min\ A(t_j, v_i)\\&\text{subject to } A(t_j, v_i) \leq S(t_j, v_i)\\&\qquad t_j^f - F(t_j,\quad v_i) \geq 0.\end{aligned} \tag{7}$$

In (7), while meeting task deadlines, we minimize the waiting time of VM i to schedule task j and thus minimize the cost to be paid by users who utilize cloud services.

The scheduling algorithm is shown in Algorithm 4.

Lines 4–6 and Lines 8–11 of Algorithm 1 require $O(n)$ time, respectively. Lines 4–18 of Algorithm 2 have double loops, requiring $O(n^2)$ time, and Line 9 in the double loops require $O(n)$ time. Therefore, Lines 4–18 of Algorithm 2 require $O(n^3)$ time. In addition, Line 3 of Algorithm 2 requires $O(n \log n)$ time. Lines 2–21 of Algorithm 3 have a single loop, requiring $O(n)$ time. Lines 3–9 of Algorithm 4 require $O(n)$ time. Lines 10–14 have a single loop, and Line 11 in the single loop requires $O(n)$ time, therefore requiring $O(n^2)$ time. Algorithms 1–4 together have $O(n) + O(n) + O(n^3) + O(n \log n) + O(n) + O(n^2)$. Hence, the overall time complexity of the proposed method is $O(n^3)$.

An example is given to show the proposed method. In Fig. 3, there are four VMs of three types. There are two type-2 VMs. Six tasks have been assigned to their corresponding VMs. Tasks t_1 and t_6 are assigned to the same type-2 VM. A newly arrived task t_7 is matched with type-2 VM.

From Fig. 3, VMs v_2 and v_3 are of type 2. So, t_7 could be assigned to one of them. We can also see that there are three idle time slots, i.e., a, b, and c. To save t_7's waiting time, we should schedule it at the earliest possibility. According to the min-time and min-cost principle, VM v_2 is more suitable than v_3 because its task t_3 starts earlier than v_3's t_4 and its execution time is shorter than the latter. Therefore, t_7 is assigned to VMv_2, as shown in Fig. 4.

In Fig. 4, according to t_7's deadline, $S(t_7, v_2)$ is 17. The idle time slot d is shown. It is a buffer time between t_3 and t_7. When t_3 at v_2 is successfully executed, t_7 can move forward, as shown in Fig. 5. In Fig. 5, t_3 has been scheduled; hence t_7 can be scheduled right after it. In other words, t_7 does not need to wait for its scheduling.

After t_j is scheduled at v_i successfully, it is added into the database of historical task scheduling, hence increasing the

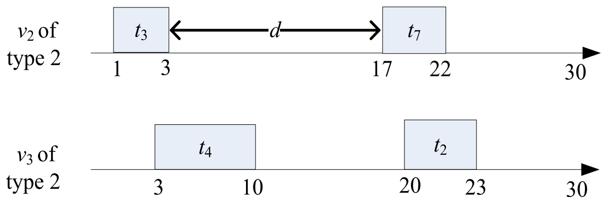


Fig. 4. Scheduling task t_7.

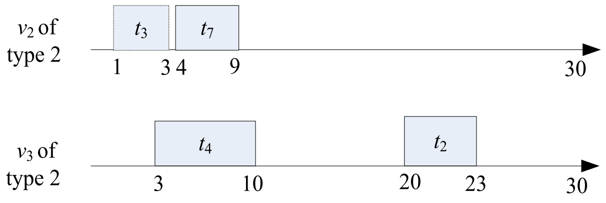


Fig. 5. Schedule adjustment for task t_7.

number of samples in the database. Expanded database can help us improve accuracy to compute the ratio of various tasks to all the tasks and then more accurately predict the types of user tasks.

IV. Experiments

A. Experimental Environment

Experimental environment includes CPU (AMD quad core, 2.4 GHz), memory (8.0 GB), HD (500G), Windows 7 operating system, Eclipse, JDK7.0, and CloudSim. As a cloud computing simulation platform, Cloudsim is widely used [10], [25]. Based on it, a dynamic scheduling process can be carried out. Our implementation inherits and extends some of the existing classes, such as *Vm*, *DatacenterBroker*, *Cloudlet*, and *Host*. The method of creating VM objects is from *Vm* in CloudSim [29] as follows:

```
Vm vm
   = newVm(vmid, brokerId, mips, pesNumber, ram,
           bw, size, vmm,
           new CloudletSchedulerTimeShared())
```

where

vmid	unique ID of the VM;
brokerId	ID of the VM's owner;
mips	mips;
pesNumber	amount of CPUs;
ram	amount of ram;
bw	amount of bandwidth;
size	amount of storage it will use, at least initially;
vmm	VM monitor;
cloudletScheduler	cloudletScheduler policy for cloudlets scheduling.

B. Experimental Results and Analysis

The effect of classifying tasks of the first stage is shown in Table IV.

Based on the above experimental configuration, we compare our method with Min–Min [20] and Max–Min [22].

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE IV
EFFECT OF CLASSIFYING TASKS OF THE FIRST STAGE

U	*TASK*	δ
VM types	Tasks of an initial task set	It marks whether there exist concrete VMs

1) Min–Min: It schedules tasks at VMs that are able to start early and perform tasks in the shortest time. Tasks are scheduled in accordance with the "earliest and shortest" principle. Min–Min algorithm needs to calculate which VM can implement tasks at the earliest and finish them fastest.
2) Max–Min: It considers task scheduling by task execution difficulty. That means the most difficult task is to be first scheduled.

Time complexity of Min–Min and Max–Min are both $O(n^3)$ [4]. Because Min–Min and Max–Min are two standard algorithms in clouds [1], many researchers adopt them as the baseline methods [1], [4], [21].

The metrics include makespan, average waiting time, failure rate of tasks, and utilization rate of VMs.

1) *Makespan:* It is total time that tasks are scheduled and completed in clouds. The smaller the makespan, the better a schedule and service quality

$$M = \frac{\min \sum_{i=1}^{N} \tau_i}{N} \tag{8}$$

where τ_i represents the completion time of task t_i in the clouds.

2) *Average Waiting Time of Tasks:* It represents performance of overall processing capacity and throughput of the clouds

$$\bar{W} = \frac{\sum_{i=1}^{N} w_i}{N} \tag{9}$$

where w_i represents waiting time of t_i.

3) *Failure Rate of Task Scheduling:* It represents stability of the clouds

$$F = \frac{\text{Number of tasks with scheduling failures}}{\text{Total number of tasks}}\% \tag{10}$$

4) *Utilization Rate of VMs:*

$$Q = \frac{\text{Time processing tasks}}{\text{Total time}}\%. \tag{11}$$

The task data set size is 1000 in our first simulation. We divide the experimental data into ten groups. Each group's data size is 100. We compute average values based on ten experimental results. Results are collected and average values are calculated. As a result, we can obtain makespan, average waiting time of tasks, failure rate of task scheduling, and utilization rate of VMs with the improved accuracy.

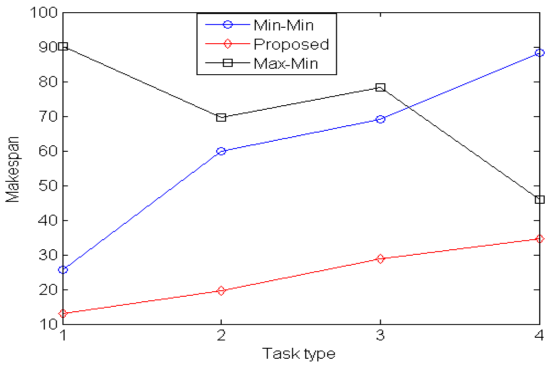


Fig. 6. Makespan versus task types.

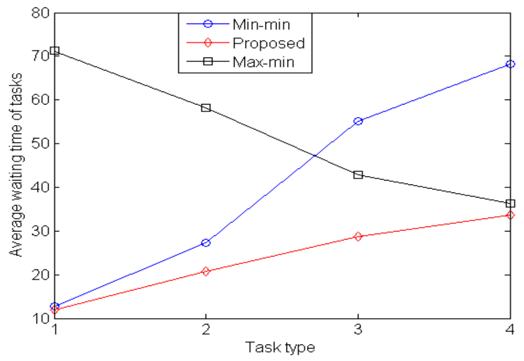


Fig. 7. Average waiting time versus task types.

Parameters of tasks and VMs are exemplified in Tables I and II, respectively. In addition, task deadline is submission time plus a fixed time (e.g., 100 ms).

In Table I, task types indicate task complexity. They also represent task scale from low to high. For example, a type-4 task needs 1000 floating point operations and 100M memory. Its deadline is far away, which is sum of its submission time and 1000 ms. They are most complex and have larger scales which need more execution time than other types of tasks.

In Table II, types 1–4 represent the performance of VMs from low to high. For example, type-4 VMs have the best performance and can provide highest CPU, memory, bandwidth, and disk memory than VMs of other types.

A VM of a higher type in Table II is suitable for running a more complex task in Table I. Otherwise, some resources of VMs are wasted. For example, if type-1 tasks are scheduled at type-4 VMs, the latter's CPU and memory resources are clearly underutilized and thus wasted. At the worst, some large tasks may be delayed or canceled because VMs of higher types are occupied when they arrive.

Makespan is shown in Fig. 6. Horizontal axis shows task types where $i \in \{1, 2, 3, 4\}$ denotes task type i.

We can see that makespan of Min–Min and the proposed methods both increase when types vary from 1 to 4, while makespan of Max–Min fluctuates according to types. Makespan of the proposed method is the smallest and it is superior to the other two methods.

Fig. 7 shows average waiting time obtained via three methods. The proposed and Min–Min methods show their upward trend of waiting time when tasks become more complex. The former has obvious advantage than the latter

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

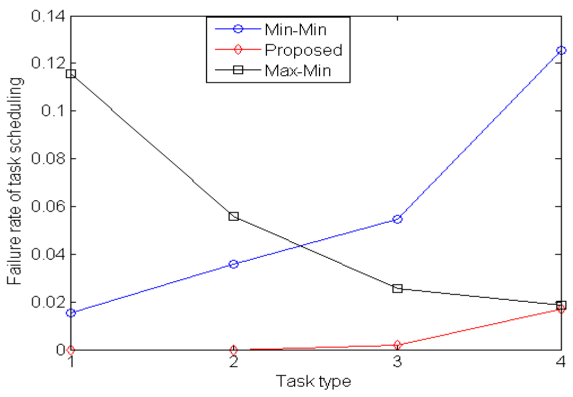


Fig. 8. Failure rate versus task types.

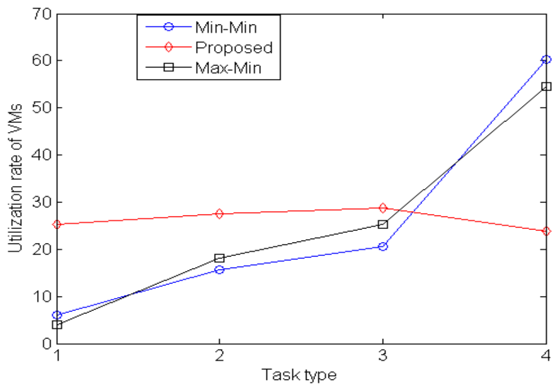


Fig. 9. Utilization rate of VMs versus task types.

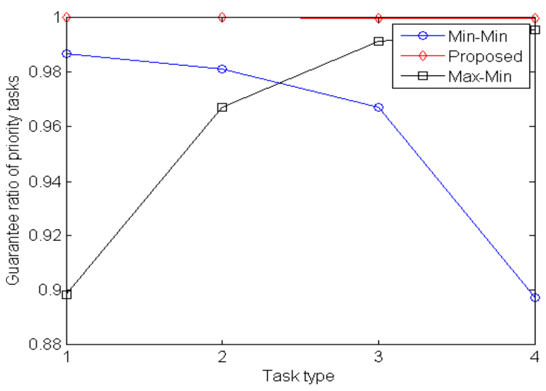


Fig. 10. Guarantee ratio of priority tasks.

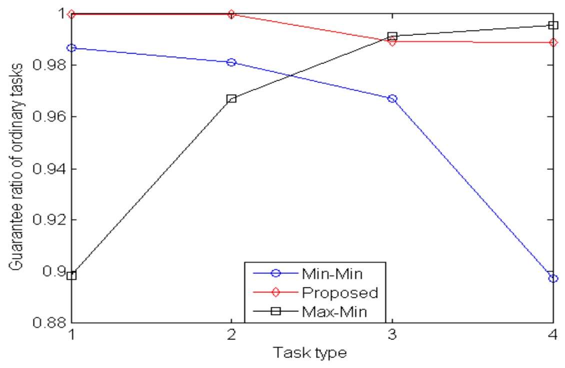


Fig. 11. Guarantee ratio of ordinary tasks versus task types.

for complex tasks, i.e., types 3 and 4. When tasks become complex, average waiting time of Max–Min is decreased. For these experiments, the proposed method well beat Max–Min for types 1 and 2 tasks. For types 3 and 4 tasks, the proposed one holds some advantage, while their difference is smaller and smaller. It is likely that Max–Min may outperform the proposed one for very complex tasks. But given a positively skewed data set, waiting time of Max–Min is to be longer than that of the proposed one for more complex tasks. A positively skewed data set means the mass of the distribution is concentrated on the right, such as task types are 1, 4, 4, 4, and 4. Assume that each type has the same task count. Because the data set has many complex tasks, then Max–Min produces poor average waiting time results.

From Fig. 8, we can see that, as task complexity increases, failure rates of the proposed method and Min–Min increase too. The difference between them is that the proposed method has a significantly smaller rate for all four task types. When task types vary from 1 to 4, Min–Min has higher and higher failure rate. Its failure rate increase sharply for type-4 tasks. For Max–Min, although failure rate decreases, it is bigger than that of the proposed one. It is better than Min–Min for types 3 and 4.

In Fig. 9, we can see that the utilization rate VMs of the proposed method is basically flat. The same disadvantage of Min–Min and Max–Min is that type-4 VMs have much higher utilization rate, while lower type VMs have lower utilization rate. Therefore, workload of the two methods is not well balanced. Especially, utilization rate of type-4 VMs of the proposed method is about 25%, while for Min–Min and Max–Min, each of their utilization rates is about 60%. Obviously, the proposed method has more advantages than Min–Min and Max–Min in workload balance.

From Fig. 10, we can see that, as task complexity increases, guarantee ratio of priority tasks with the proposed method is close to 1. For Min–Min, the ratio decreases from nearly 1 to about 0.9, while for Max–Min, the ratio increases from about 0.9 to nearly 1.

As shown in Fig. 11, the guarantee ratio of ordinary tasks with the proposed method is close to 1 for types 1 and 2 tasks. It decreases from 1 to about 0.95 for types 3 and 4 tasks. As task complexity increases, guarantee ratios of ordinary tasks of Min–Min and Max–Min are almost opposite to each other.

In Fig. 12, the guarantee ratio of all tasks of the proposed method is close to 1 for all task types. While for Min–Min and Max–Min, they have upward and downward trends as task complexity increases, respectively. From Figs. 10–12, we can conclude that the proposed method has better guarantee ratio than Min–Min and Max–Min do.

Our simulation also uses the Google Cluster Trace, available at https://github.com/google/cluster-data. It includes 10 000 actual workflow traces. We perform the experiments for three methods and obtain the results as plotted in Figs. 13–17 based on the modified dataset. The task types are shown in Table V.

From Table V, we can see that there are mainly two kinds of tasks. One is computing tasks which need high CPU and low memory. The other is storing tasks which need low CPU and

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

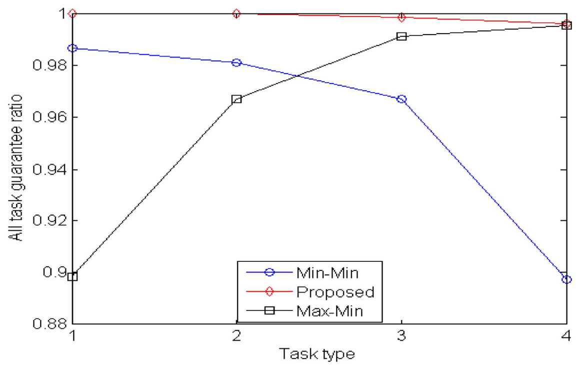


Fig. 12. Guarantee ratio of all tasks versus task types.

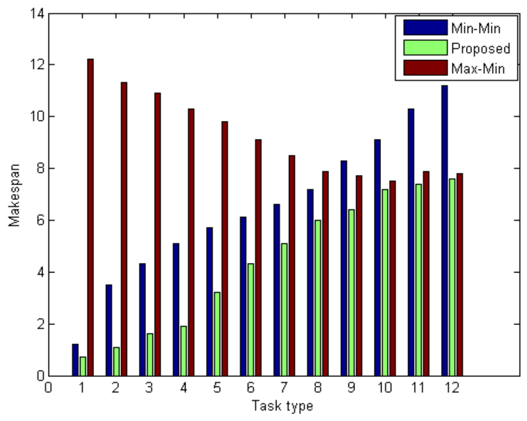


Fig. 13. Makespan versus task types.

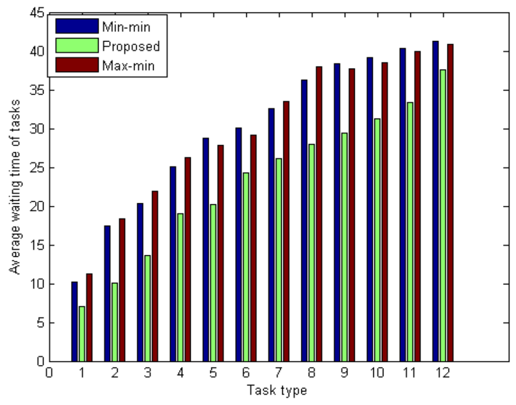


Fig. 14. Average waiting time versus task types.

high memory. In clouds, the network bandwidth and hard disk are enough which are not considered in the part. Accordingly, the VM types are shown in Table VI.

Table VI lists the parameters that are the critical ones of all VMs. Because of fiber optic network and flexible hard disk provision in clouds, network bandwidth and hard disk are not critical and thus treated as having perfect match with all the tasks.

From Figs. 13–17, we can see that the proposed technique can scale well to large instances. In Fig. 13, we can conclude that: 1) the makespans of Min–Min and the proposed schedules both increase when task types vary from 1 to 12, while that of Max–Min's decreases and 2) the proposed method can achieve the smallest makespan in comparison with its two peers.

TABLE V
TASK TYPES AND THEIR PARAMETERS

Task types	Count	CPU demand(GHz)	Memory demand(GB)	Duration(10^6s)
1	2.2*10^6	0.02-0.03	0.009-0.013	0.25-0.5
2	1.0*10^4	0.25-0.32	0.02-0.09	0.65-0.75
3	5.0*10^5	0.02-0.05	0.15-0.17	0.45-0.65
4	0.9*10	0.06-0.14	0.8-0.85	0.60-0.80
5	6.5*10^3	0.01-0.04	0.03-0.09	1.60-2.10
6	2.0*10^6	0.02-0.07	0.3-0.38	1.50-2.10
7	2.5^10^4	0.16-0.17	0.02-0.12	1.65-2.15
8	6.5*10^5	0.02-0.10	0.002-0.015	0.65-1.05
9	2.0*10^6	0.04-0.08	0.03-0.07	1.65-2.05
10	1.0^10^7	0.02-0.05	0.002-0.017	1.20-1.57
11	1.0*10^6	0.08-0.10	0.002-0.008	0.95-1.40
12	0.9*10^6	0.02-0.06	0.002-0.011	1.85-2.35

TABLE VI
VM TYPES AND THEIR PARAMETERS

VM types	CPU(GHz)	Memory(GB)
1	0.03	0.013
2	0.32	0.09
3	0.05	0.17
4	0.14	0.85
5	0.04	0.09
6	0.07	0.38
7	0.17	0.12
8	0.1	0.015
9	0.08	0.07
10	0.05	0.017
11	0.1	0.008
12	0.06	0.011

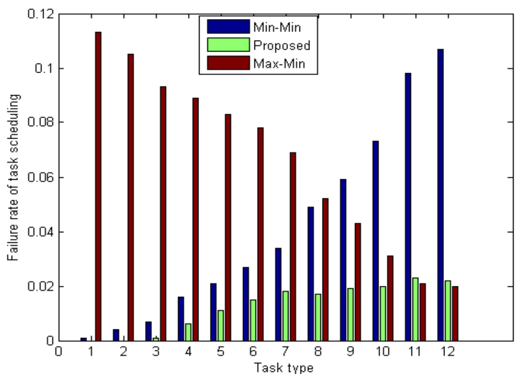


Fig. 15. Failure rate versus task types.

Fig. 14 shows average waiting time obtained via three methods. The waiting time increases for three methods when tasks become more complex. The proposed method has the smallest average waiting time among three methods for all task

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

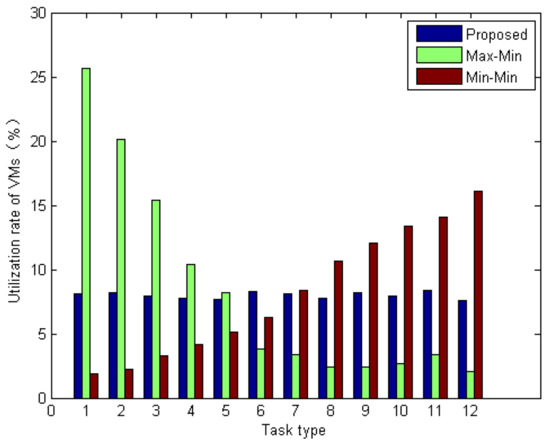


Fig. 16. Utilization rate of VMs versus task types.

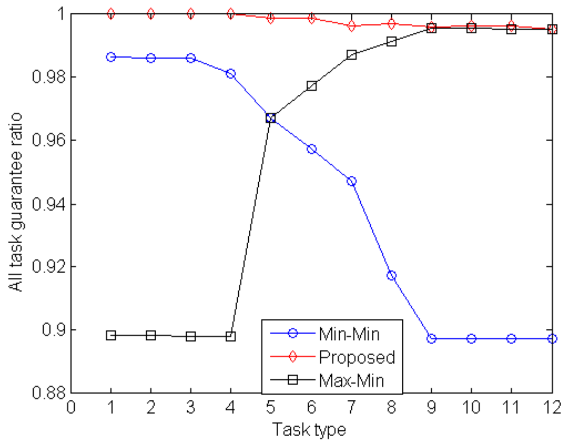


Fig. 17. Guarantee ratio of all tasks versus task types.

types. Fig. 15 suggests that the proposed method can achieve the lowest failure rate. Fig. 16 shows that the utilization rate of VMs is flat when the proposed method is used, thus implying very balanced workload among all VMs. The disadvantage of Min–Min is that higher type VMs have much higher utilization rate, while lower type VMs have lower utilization rate. The disadvantage of Max–Min is that higher type VMs have much lower utilization rate, while lower type VMs have very high utilization rate. Clearly, the workload of Min–Min and Max–Min are not well balanced. Hence, the proposed method is superior to its peers in terms of workload balancing. Fig. 17 shows that the proposed method can deliver the highest guarantee ratio of all tasks among three methods.

From the above experimental result analysis, we can conclude the following.

1) For tasks of different types, the proposed method has its obvious advantage. When the number and scale of tasks continue to increase, the proposed method increases makespan and average waiting time more slowly than Min–Min and Max–Min do.
2) Failure rate of the proposed method is lower than those of its two peers.
3) Workload of VMs of each VM type as obtained by the proposed method is better balanced than that by its peers.
4) The proposed method offers higher guarantee ratios of priority and ordinary tasks than its peers do.

V. Conclusion

In this paper, we propose a two-stage task scheduling framework and its related algorithms to achieve desired task scheduling and execution and improve service quality of the clouds. Based on historical task scheduling information, a proper number of VMs with different resource attributes are precreated. It can save much time to create VMs and decrease the failure rate of task scheduling. According to task complexity, most suitable VMs are chosen from the precreated ones to process tasks. Based on the two-stage strategy, the task scheduling algorithms are proposed. Experimental results show that they can improve the performance of task scheduling and execution in comparison with well-known Min–Min [20] and Max–Min [22] methods. Further research work includes deploying the method to an actual cloud computing system to test its performance and improving the proposed scheduling method by considering how to minimize the energy consumption while guaranteeing the service quality [30]–[37].

References

[1] S. K. Panda and P. K. Jana, "Efficient task scheduling algorithms for heterogeneous multi-cloud environment," *J. Supercomput.*, vol. 71, no. 4, pp. 1505–1533, Apr. 2015.
[2] Z. H. Zhan, X. F. Liu, Y. J. Gong, J. Zhang, H. S. H. Chung, and Y. Li, "Cloud computing resource scheduling and a survey of its evolutionary approaches," *ACM Comput. Surv.*, vol. 47, no. 4, pp. 1–33, Jul. 2015.
[3] X. Zuo, G. Zhang, and W. Tan, "Self-adaptive learning PSO-based deadline constrained task scheduling for hybrid IaaS cloud," *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 2, pp. 564–573, Apr. 2014.
[4] R. Duan, R. Prodan, and X. Li, "Multi-objective game theoretic schedulingof bag-of-tasks workflows on hybrid clouds," *IEEE Trans. Cloud Comput.*, vol. 2, no. 1, pp. 29–49, Jan. 2014.
[5] S. Wang, Z. Liu, Q. Sun, H. Zou, and F. Yang, "Towards an accurate evaluation of quality of cloud service in service-oriented cloud computing," *J. Intell. Manuf.*, vol. 25, no. 2, pp. 283–291, Apr. 2014.
[6] J. W. Lin, C. H. Chen, and C. Y. Lin, "Integrating QoS awareness with virtualization in cloud computing systems for delay-sensitive applications," *Future Generat. Comput. Syst.*, vol. 37, pp. 478–487, Jul. 2014.
[7] M. A. Rodriguez and R. Buyya, "Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds," *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 222–235, Apr.-Jun.2014.
[8] C. Delimitrou, D. Sanchez, and C. Kozyrakis, "Tarcil: Reconciling scheduling speed and quality in large shared clusters," in *Proc. SoCC*, Aug. 2015, pp. 97–110.
[9] C. Delimitrou and C. Kozyrakis, "QoS-aware scheduling in heterogeneous datacenters with paragon," *ACM Trans. Comput. Syst.*, vol. 31, no. 4, pp. 1–34, Dec. 2013.
[10] Y. Xia, M. Zhou, X. Luo, J. Li, Y. Huang, and Q. Zhu, "Stochastic modeling and quality evaluation of infrastructure-as-a-service clouds," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 1, pp. 162–170, Jan. 2015.
[11] H. Yuan, J. Bi, W. Tian, M. C. Zhou, B. H. Li, and J. Li, "TTSA: An effective scheduling approach for delay bounded tasks in hybrid clouds," *IEEE Trans. Cybern.*, to be published, doi: 10.1109/TCYB.2016.2574766.
[12] N. Jain, I. Menache, J. Naor, and J. Yaniv, "Near-optimal scheduling mechanisms for deadline-sensitive jobs in large computing clusters," *ACM Trans. Parallel Comput.*, vol. 2, no. 1, pp. 1–29, Apr. 2015.
[13] Y. W. Chen and J. M. Chang, "EMaaS: Cloud-based energy management service for distributed renewable energy integration," *IEEE Trans. Smart Grid*, vol. 6, no. 6, pp. 2816–2824, Jul. 2015.
[14] K. Wu, P. Lu, and Z. Zhu, "Distributed online scheduling and routing of multicast-oriented tasks for profit-driven cloud computing," *IEEE Commun. Lett.*, vol. 20, no. 4, pp. 684–687, Apr. 2016.
[15] S. K. Garg, C. S. Yeo, A. Anandasivam, and R. Buyya, "Environment-conscious scheduling of HPC applications on distributed cloud-oriented data centers," *J. Parallel Distrib. Comput.*, vol. 71, no. 6, pp. 732–749, Jun. 2011.
[16] X. Zhu, L. T. Yang, H. Chen, J. Wang, S. Yin, and X. Liu, "Real-time tasks oriented energy-aware scheduling in virtualized clouds," *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 168–180, Apr. 2014.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

[17] F. Pop, V. Cristea, N. Bessis, and S. Sotiriadis, "Reputation guided genetic scheduling algorithm for independent tasks in inter-clouds environments," in *Proc. AINA*, Barcelona, Spain, Mar. 2013, pp. 772–776.

[18] M. A. Tawfeek, A. El-Sisi, A. E. Keshk, and F. A. Torkey, "Cloud task scheduling based on ant colony optimization," in *Proc. ICCES*, Chennai, India, Nov. 2013, pp. 64–69.

[19] J. Li, M. Qiu, J. Niu, Z. Zong, W. Gao, and X. Qin, "Feedback dynamic algorithms for preemptable job scheduling in cloud systems," in *Proc. WI/IAT*, Aug. 2010, pp. 561–564.

[20] J. Li, M. Qiu, Z. Ming, G. Quan, X. Qin, and Z. Gu, "Online optimization for scheduling preemptable tasks on IaaS cloud systems," *J. Parallel Distrib. Comput.*, vol. 72, no. 5, pp. 666–677, May 2012.

[21] S. K. Panda and P. K. Jana, "An efficient task scheduling algorithm for heterogeneous multi-cloud environment," in *Proc. Int. Conf. Adv. Comput., Commun. Inform. (ICACCI)*, Sep. 2014, pp. 1204–1209.

[22] S. Devipriya and C. Ramesh, "Improved Max-min heuristic model for task scheduling in cloud," in *Proc. ICGCE*, Chennal, India, Dec. 2013, pp. 883–888.

[23] E. Santos-Neto, W. Cirne, F. Brasileiro, and A. Lima, "Exploiting replication and data reuse to efficiently schedule data-intensive applications on grids," in *Job Scheduling Strategies for Parallel Processing* (Lecture Notes in Computer Science), Berlin, Germany: Springer-Verlag, vol. 3277, Jun. 2005, pp. 210–232.

[24] H. Chen, F. Wang, N. Helian, and G. Akanmu, "User-priority guided Min-Min scheduling algorithm for load balancing in cloud computing," in *Proc. PARCOMPTECH*, Bangalore, India, Feb. 2013, pp. 1–8.

[25] J. Wang, W. Bao, X. Zhu, L. T. Yang, and Y. Xiang, "FESTAL: Fault-tolerant elastic scheduling algorithm for real-time tasks in virtualized clouds," *IEEE Trans. Comput.*, vol. 64, no. 9, pp. 2545–2558, Sep. 2015.

[26] C.-Y. Chen, "Task scheduling for maximizing performance and reliability considering fault recovery in heterogeneous distributed systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 521–532, Feb. 2016.

[27] X. Zhu, C. Chen, L. T. Yang, and Y. Xiang, "ANGEL: Agent-based scheduling for real-time tasks in virtualized clouds," *IEEE Trans. Comput.*, vol. 64, no. 12, pp. 3389–3403, Dec. 2015.

[28] T. He, S. Chen, H. Kim, L. Tong, and K. W. Lee, "Scheduling parallel tasks onto opportunistically available cloud resources," in *Proc. CLOUD*, Honolulu, HI, USA, Jun. 2012, pp. 180–187.

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

[30] M. M. Hassan, M. Abdullah-Al-Wadud, and G. Fortino, "A socially optimal resource and revenue sharing mechanism in cloud federations," in *Proc. 19th IEEE Int. Conf. Comput. Supported Cooperat. Work Design*, Calabria, Italy, May 2015, pp. 620–625.

[31] G. Fortino, A. Guerrieri, W. Russo, and C. Savaglio, "Integration of agent-based and cloud computing for the smart objects-oriented IoT," in *Proc. 18th IEEE Int. Conf. Comput. Supported Cooperat. Work Design*, Hsinchu, Taiwan, May 2014, pp. 493–498.

[32] J. Bi *et al.*, "Application-aware dynamic fine-grained resource provisioning in a virtualized cloud data center," *IEEE Trans. Autom. Sci. Eng.*, vol. 14, no. 2, pp. 1172–1184, Apr. 2017.

[33] H. Yuan, J. Bi, W. Tan, and B. H. Li, "CAWSAC: Cost-aware workload scheduling and admission control for distributed cloud data centers," *IEEE Trans. Autom. Sci. Eng.*, vol. 13, no. 2, pp. 976–985, Apr. 2016.

[34] W. Zheng *et al.*, "Percentile performance estimation of unreliable IaaS clouds and their cost-optimal capacity decision," *IEEE ACCESS*, vol. 5, pp. 2808–2818, Mar. 2017.

[35] M. H. Ghahramani, M. C. Zhou, and C. T. Hon, "Toward cloud computing QoS architecture: Analysis of cloud systems and cloud services," *IEEE/CAA J. Autom. Sinica*, vol. 4, no. 1, pp. 5–17, Jan. 2017.

[36] Y. Cai, Z. Tang, Y. Ding, and B. Qian, "Theory and application of multi-robot service-oriented architecture," *IEEE/CAA J. Autom. Sinica*, vol. 3, no. 1, pp. 15–25, Jan. 2017.

[37] P. Xiong, C. Pu, Z. Wang, and G. Jung, "Control-based approaches to dynamic resource management in cloud computing," in *Contemporary Issues in Systems Science and Engineering*, M. C. Zhou, H. X. Li, and M. Weijnen. Hoboken, NJ, USA: Wiley, 2015, pp. 601–617.

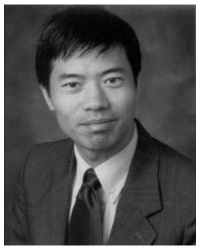
[38] J. Li, J. Zhang, C. Jiang and M. C. Zhou, "Composite particle swarm optimizer with historical memory for function optimization," *IEEE Trans. Cybern.*, vol. 45, no. 10, pp. 2350–2363, Oct. 2015.

[39] Q. Kang, M. C. Zhou, J. An, and Q. Wu, "Swarm intelligence approaches to optimal power flow problem with distributed generator failures in power networks," *IEEE Trans. Autom. Sci. Eng.*, vol. 10, no. 2, pp. 343–353, Apr. 2013.

[40] W. Han, J. Xu, M. C. Zhou, G. Tian, P. Wang, X. Shen and S.-H. E. Hou, "Cuckoo-search and particle-filter-based inversing approach to estimating defects via magnetic flux leakage signals," *IEEE Trans. Magn.*, vol. 52, no. 4, Apr. 2016, Art. no. 6200511.

PeiYun Zhang (M'16) received the B.S. degree from Anhui Normal University, Wuhu, China, in 1998, the M.S. degree from Northwest University, Xi'an, China, in 2005, and the Ph.D. degree from the School of Computer Science and Technology, Nanjing University of Science and Technology, Nanjing, China, in 2008.

She was a Post-Doctoral Researcher with the University of Science and Technology China, Hefei, China, from 2010 to 2013, and a Visiting Scholar with the New Jersey Institute of Technology, Newark, NJ, USA, in 2016. She is currently a Professor with the School of Mathematics and Computer Science, Anhui Normal University. Her research interests include cloud computing, big data, trust computing, Petri nets, web service, and intelligent information processing. She has published over 30 papers in the above areas.



MengChu Zhou (S'88–M'90–SM'93–F'03) received the B.S. degree in control engineering from the Nanjing University of Science and Technology, Nanjing, China, in 1983, the M.S. degree in automatic control from the Beijing Institute of Technology, Beijing, China, in 1986, and the Ph.D. degree in computer and systems engineering from the Rensselaer Polytechnic Institute, Troy, NY, USA, in 1990.

He joined the New Jersey Institute of Technology (NJIT), Newark, NJ, USA, in 1990, and is now a Distinguished Professor of Electrical and Computer Engineering. He has over 690 publications including 12 books, 370+ journal papers (over 270 in the IEEE TRANSACTIONS), and 28 book-chapters. His research interests include Petri nets, intelligent automation, Internet of Things, big data, and intelligent transportation.

Dr. Zhou was a recipient of the NSF's Research Initiation Award, the CIM University-LEAD Award from the Society of Manufacturing Engineers, the Perlis Research Award and the Fenster Innovation in Engineering Education Award from NJIT, the Humboldt Research Award for U.S. Senior Scientists, the Leadership Award and the Academic Achievement Award from the Chinese Association for Science and Technology–USA, the Distinguished Lecturership, the Franklin V. Taylor Memorial Award and the Norbert Wiener Award from the IEEE SMC Society, and the Distinguished Service Award from the IEEE Robotics and Automation Society. He has been among most highly cited scholars for years and ranked top one in the field of engineering worldwide in 2012 by the Web of Science/Thomson Reuters. He is a Life Member of the Chinese Association for Science and Technology-USA, and served as its President in 1999. He is a Fellow with the International Federation of Automatic Control (IFAC) and the American Association for the Advancement of Science (AAAS).