Mobile Networks and Applications (2020) 25:690–700
https://doi.org/10.1007/s11036-019-01450-0

A Novel Approach to Scheduling Workflows Upon Cloud Resources with Fluctuating Performance

Yi Pan[1] · Shu Wang[2] · Lei Wu[1] · Yunni Xia[3] · Wanbo Zheng[4] · Shanchen Pang[5] · Ziyang Zeng[6] · Peng Chen[7] · Yawen Li[3]

Published online: 16 January 2020

Abstract
Cloud computing is recently getting increasingly popular for supporting scientific applications and complex business processes. Clouds are highly potent for executing workflow-based tasks due to the fact that they provide elastic resource provisioning styles through which computational-intensive workflows can obtain requested resources according to their elastic demand and establish execution environment over virtual machines (VMs). However, it remains a challenge to guarantee cost-effectiveness and quality of service of workflow deployed upon clouds due to the fact that real-world cloud infrastructures are usually with fluctuating and time-varying performance. Existing researches mainly consider that cloud infrastructures are with fixed, random, or bounded quality of service (QoS). In this work, however, we consider that scientific computing processes to be supported by decentralized cloud infrastructures with fluctuating QoS and aim at managing the monetary cost of workflows with the completion-time constraint to be satisfied. We address the performance-variation-aware workflow scheduling problem by leveraging a time-series-based prediction model and a Critical-Path-Duration-Estimation-based (CPDE for short) VM Selection strategy. The proposed method is capable of exploiting real-time trends of performance changes of cloud infrastructures and generating dynamic workflow scheduling plans. To prove the effectiveness of our proposed method, we perform extensive experimental case analysis over real-world third-party commercial clouds and show that our method clearly beats existing approaches.

Keywords Cloud computing · Scheduling · Workflow · Quality-of-service

1 Introduction

The cloud paradigm is becoming increasingly popular in supporting versatile types of industrial, scientific, and social applications [1–4]. It provides tools and techniques for designing computational-intensive applications that are more cost-effective than traditional parallel computing techniques. The cloud management process only distributes demanded resources to resource requesters and in such a way the resource utilization and monetary cost of invoked cloud resources can be optimized. According to the provisioning styles and architectural patterns, main-stream cloud computing systems can fall into the following categories: infrastructure clouds (IaaS), platform clouds (PaaS), and software clouds (SaaS). The IaaS model provisions resources through instances of virtual machine (VM) created in data centers or server nodes. It is shown [5–9] that IaaS clouds are highly potent and effective in supporting workflow-based applications. Workflows are usually instantiated by carrying out the following steps [10–12] :1) VMs instances are invoked to execute specific workflow tasks; 2) a schedule is yielded and then the decided task-resource mapping is performed.

In recent years, the optimal scheduling issues of workflows attracted wide attention [13–18]. A major obstacle for ensuring user-end QoS of cloud workflows is that real-time cloud infrastructures and VMs are subject to performance fluctuations and variations. Schad et al. [19] showed that virtual machine (VM) performance in the Amazon EC2 could degrade by 23% when cloud nodes are under stress. Jakson et al. [20] observed that cloud performance can vary by 30-65% when inter-cloud-node transfer were under stress. Run-time changes of the QoS of cloud infrastructures potentially impact the

✉ Lei Wu
wulei@uestc.edu.cn

Extended author information available on the last page of the article.

user-experienced QoS of workflow-oriented applications executed upon cloud infrastructures and further leads to violations to Service-Level-Agreement (SLA) [21]. Note that performance fluctuations of cloud infrastructures may bring in extra operational cost due to the fact that migrations or fault-handling activities are needed to counter such performance degradations when SLA constraints are breached. However, a careful literature review shows that most existing works ignored such real-time performance fluctuations and addressed the scheduling problem by considering static, stochastic, or bounded QoS data. However, our work suggests that, these traditional scheduling methods can show unsatisfactory performance of scheduled workflows when the supporting clouds are with time-varying performance.

To address the above concern, we proposes a novel method that combines the ability of a time-series-based prediction model and a critical-path-duration-estimation-based VM selection strategy. Our method is able to exploit real-time performance variations of VMs and appropriately schedule workflow tasks by considering and optimizing the execution durations of the critical-path. The proposed method aims at reducing the monetary cost of workflows with the constraint of workflow completion time. Extensive experimental studies up on real-world public clouds, i.e., Huawei, Baidu, and Tencent cloud services and multiple workflow templates clearly show that our method outperform its peers in terms of workflow completion times, cost, and SLA violation rates (Fig. 1).

2 Related work

It is generally accepted that the problem of scheduling multi-task workflows upon multiple distributed resources is NP-hard [22]. In this view, heuristic and meta-heuristic methods can be used to generated high-quality and approximate solutions with polynomial time-complexity.

For example, Casas et al. [23] considered an bio-inspired framework with the Efficient Tune-In (GA-ETI) mechanism for scientific processes executed over cloud platforms. It optimizes both workflow performance and cost. Verma et al. [24] presented a non-dominated-sorting-based Hybrid Particle-Swarm-Optimization (HPSO) approach. It optimizes both cost and performance. Zhou et al. [25] developed a fuzzy-dominance-sort-based earliest-finish-time (FDHEFT) algorithm. It minimizes both cost and response time of applications over heterogeneous cloud infrastructures. Wang et al. [26] developed a multi-objective game-theoretic scheduling framework. The scheduling scheme is generated by multi-stage dynamic game to achieve the approximate optimal solution under multi-objective constraints. A major limitation of these works lies in that they consider time-invariant and non-fluctuating performance of cloud services and infrastructures. These methods can be ineffective when supporting cloud services and infrastructures are with time-varying performance.

Recently, many dynamic workflow scheduling methods are proposed, where performance of cloud services and infrastructures are no longer considered to be invariant. For

Fig. 1 Notaion and meaning

Symbols	Description
C	The cost of executing a workflow
D	The deadline constraint of a workflow
T	The set of tasks of a workflow
K	The set of VM types
n	The number of tasks
$*t_i$	The preceding sets of t_i
t_i*	The subsequent sets of t_i
t_i	The i^{th} task of a workflow
x_{ij}	The data transfer time between t_i and t_j
v_j	The j^{th} VM instance
$w(i)$	A function to indicate the VM to which task t_i is to be scheduled
$g(i)$	A function to indicate the type of v_j
$h(j)$	A function to indicate cost-per-unit-time of VM v_j
$EIT(j)$	A function to indicate expected idle time of VM v_j
$AIT(j)$	A function to indicate actual idle time of VM v_j
$EST(i,j)$	A function to indicate estimated starting time of task t_i if it's scheduled into VM v_j
$EFT(i,j)$	A function to indicate estimated finishing time of task t_i if it's scheduled into VM v_j
$AST(i,j)$	A function to indicate actual starting time of task t_i if it's scheduled into VM v_j
$AFT(i,j)$	A function to indicate actual finishing time of task t_i if it's scheduled into VM v_j
$EET(i,j,\theta)$	A function to indicate estimated execution time of task t_i if it's scheduled into VM v_j on condition that its execution begins at time θ
$AET(i,j,\theta)$	A function to indicate acutual execution time of task t_i if it's scheduled into VM v_j on condition that its execution begins at time θ
$ET_{cp}(i,j,\theta)$	A function to indicate estimated execution time of the critical path starting from t_i if it's scheduled into VM v_j on condition that its execution begins at time θ

example, Sahni et al. [27] proposed a cost minimization method for dynamic multi-workflow scheduling with the constraint of completion time. Considering the fluctuation of performance, the timely scheduling scheme is generated dynamically in the process of task execution. However, it generates the scheduling scheme dynamically according to the current state, and does not consider the future performance fluctuation when making the decision, so it cannot avoid choosing the virtual machine with declining trend of performance.

Mao et al. [28] developed a VM-consolidation-based algorithm for cloud-oriented workflows. It finds optimal scheduling plans to consolidate distributed VMs and assumes a given bound of response time variations. Calheiros et al. [29] considered soft deadlines and proposed a critical path identification method that utilizes idle time-slots for the improvement of resource utilization. They considered a given bound of performance variations of VMs. Poola et al. [30] considered a fault-tolerant and performance-variation-aware workflow scheduling method. Ghosh et al. [31, 32] assumed VM response time to be with given stochastic distribution, i.e., an exponential distribution. While Zheng et al. [33] assumed a Pareto distribution as an approximation.

Assuming stochastic or bounded performance of cloud services and infrastructures helps to reduce SLA violation rate. Nevertheless, as discussed in [34], such assumption may sometimes cause pessimistic estimation of system status and further lead to low resource utilization due to the fact that they tended to employ the lower bounds of performance and higher bounds of as the inputs of scheduling algorithms. Consider an inexpensive VM with unstable QoS performance and with averaged/highest response time of 14s/12s and another costly but stable one with averaged/highest execution time of 6s/6.1s. If 12s is assumed to be the threshold, the bound-based scheduling strategy probably selects the expensive one to avoid SLA violation. However, 12s occurs only in extreme cases and a smart strategy should decide the future probability of breaching the 12s constraint and choose the cheap VM when such probability is low.

3 Model description

A workflow $W = (T, E)$ is described by a Directed-Acyclic-Graph (DAG) where $T = \{t_1, t_2, \ldots, t_m\}$ denotes the set of tasks and E denotes the set of edges. The edge $e_{ij} \in E$ indicates the constraint between t_i and $t_j (i \neq j)$. t_i is the father node of t_j and t_j is the child node of t_i. This means that task t_j can only be started after t_i is completed. D identifies the pre-specified constraint.

3.1 Cloud resource model

VMs are created in the provider resource pool and vary in computing performance and price. If task t_i and t_j are executed on different virtual machines, and $e_{ij} \in E$, a transmission of data between these two VMs is inevitable. If t_i and t_j are executed on the same VM, however such transmission is unnecessary and thus the transmission time $x_{ij} = 0$.

The pricing plan is based on the pay-as-you-go billing style. Resource providers charge users according to the time they spend for occupying VMs.

3.2 Workflow computing model

The execution sequence of a workflow is usually expressed by associating an index to every task. The index is usually with the rage from 1 to m and the ith item identifies the order of running t_i. The above order relation can be decided by the function $l : T \rightarrow N^+$ and encoded as a vector in terms of a permutation of 1 to n. If i precedes k in the execution sequence, it doesn't necessarily mean that t_i precedes t_k unless they belong to the the same virtual machine. A task can be executed only when its supporting VM is ready and all its preceding tasks are finished.

If t_k follows t_i through an edge $e_{i,k}$ and they belong to different VMs, a transfer time $x_{i,k}$ is inevitable because inter-VM communication is needed. Otherwise, $x_{i,k} = 0$ in case that both tasks belong to the same VM.

Workflow tasks supported by different types of VMs show time-fluctuating performance. Moreover, a task executed by the same VM at different time shows changing performance as well. In order to well describe the dynamics of such variations at real-time and make accurate prediction, we incorporate a time-series-based prediction model as described in the next section (Fig. 2).

4 ARIMA model

Time series refers to the sequence of the values of the same statistical indicators in order of their occurrence. A time series can usually be expressed as a series of data points indexed (or listed or graphed) in time order. Time series data essentially reflects the trend of a random variable or some random variable changing with time, and the core of time series prediction method is to mine this law from the data and use it to estimate the future data. We incorporate an ARIMA time series model [35] as the underlying prediction method for processing time-fluctuating performance data of cloud infrastructures.

A time series is mathematically stationary only when its residuals are statistically independent of each other and

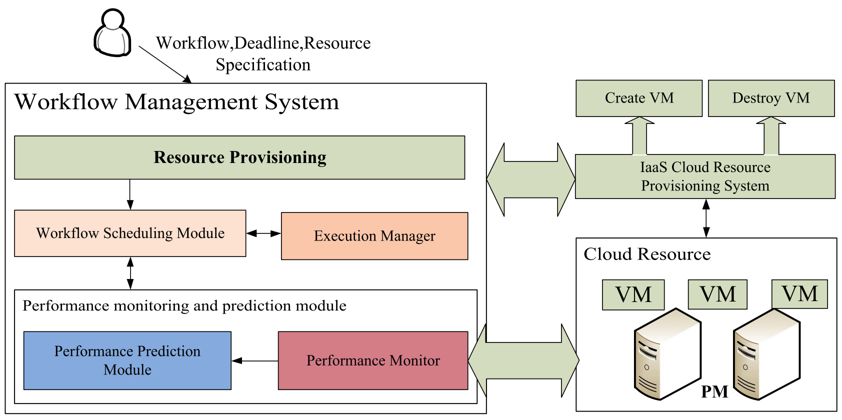

Workflow,Deadline,Resource Specification
Workflow Management System
Resource Provisioning
Workflow Scheduling Module
Execution Manager
Performance monitoring and prediction module
Performance Prediction Module
Performance Monitor
Create VM
Destroy VM
IaaS Cloud Resource Provisioning System
Cloud Resource
VM
VM
VM
PM

Fig. 2 Deploying and executing scientific task over cloud infrastructures

are constant in terms of mean and variance. Non-stationary time series can be described by an ARIMA model, provided that it turns into a stationary one after a limited number of differentiation operations.

For a non-stationary time series $\{x_t\}$, its first-order difference is:

$$\nabla x_t = x_t - x_{t-1} = x_t - Bx_t = (1-B)x_t \tag{1}$$

where B is the backshift operator. If the new series of ∇x_t is still non-stationary, more differentiations are needed until higher-order series of differences, i.e., $\nabla^d x_t$ is stationary:

$$\nabla^d x_t = \nabla^d x_t - \nabla^d x_{t-1} == (1-B)^d x_t \tag{2}$$

Then, we feed $\nabla^d x_t$ into an ARIMA model with orders p and q, denoted by ARIMA(p, q). An ARIMA model combines an autoregressive (AR) component and a moving-average one.

$$\phi(B)x_t = \theta(B)z_t \tag{3}$$

$\{z_t\}$ indicates a series of errors, $\phi(B)$ the autoregressive polynomial with order p and $\theta(B)$ the average moving polynomial with order q:

$$\phi(B) = (1 - \phi_1 B - \phi_2 B^2 - \cdots - \phi_p B^p) \tag{4}$$

$$\theta(B) = (1 + \theta_1 B + \theta_2 B^2 + \cdots + \theta_q B^q) \tag{5}$$

The non-stationary characteristic of an ARIMA series can thus be described by using a generalized autoregressive operator $\varphi(B)$:

$$\varphi(B) = \phi(B)(1-B)^d \tag{6}$$

The predicted future values can thus be defined as:

$$\varphi(B)x_t = \phi(B)(1-B)^d x_t = \theta(B)z_t \tag{7}$$

So

$$\phi(B)\omega_t = \theta(B)z_t \tag{8}$$

where

$$\omega_t = (1-B)^d x_t = \nabla^d x_t \tag{9}$$

The predicted values will be input into the scheduling algorithm to generate the scheduling scheme.

In this work, we use a benchmark tool to measure the real-time performance, in terms of execution time, and use the ARIMA model to obtain the predicted values as shown in the Figs. 3, 4 and 5. Generating prime numbers in the specified range is defined as an event in the benchmark environment (from https://dev.mysql.com/downloads/benchmarks.html).

5 Symbol definition and fomulation

Cost-effectiveness and performance are usually contradicting objectives of cloud-oriented workflow scheduling. Our proposed method aims at reconciling them and yield a cost-effective schedule with minimal cost while satisfying performance constraints. The underlying problem can therefore be formulated as below, where C denotes the cost of

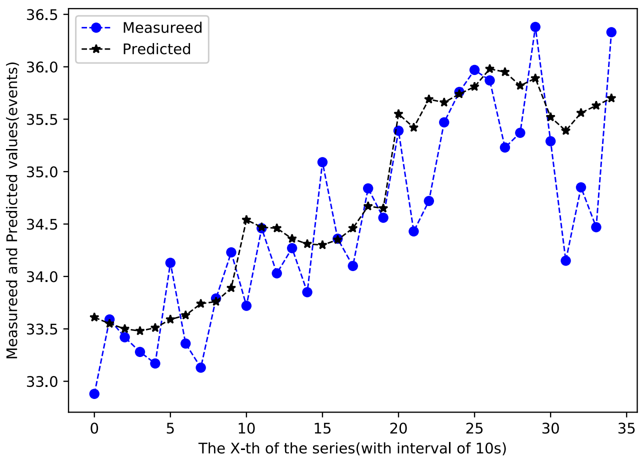


Fig. 3 Measured and predicted events of generating prime numbers on Baidu cloud

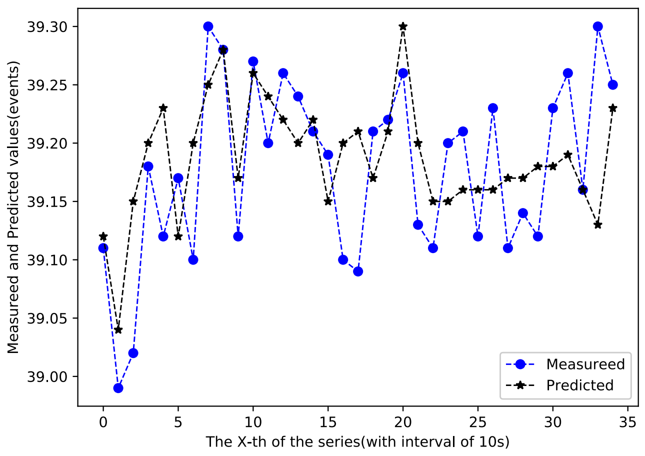


Fig. 4 Measured and predicted events of generateing prime numbers on Huawei cloud

executing a scientific workflow, D denotes the deadline from SLA, $h(j)$ denotes the cost-per-time for using the VM v_j.

$$\min C = \sum_{i=1}^{|n|}((AFT(i, w(i)) - AST(i, w(i))) * h(w(i)))$$
$$subject\ to\ \max AFT(i, w(i)) < D \tag{10}$$

The other formulas and explanations are as follows:

$EST(i, j)$ is the estimated starting time of task t_i if it is scheduled into VM v_j, calculated by the completion time of the parent tasks, the transmission delay, and the estimated idle time of the virtual machine. If the task has no parents, it can start executing when the virtual machine and runtime environment are available.

$$EST(i, m) = \begin{cases} \max\limits_{\substack{t_j \in {}^*t_i \\ n \neq m}} (EFT(j, n) + x_{j_i}, EFT(j, m), EIT(m)) & \text{if } {}^*t_i \neq \emptyset \\ EIT(m) & \text{if } {}^*t_i = \emptyset \end{cases} \tag{11}$$

$AST(i, j)$ indicates the actual starting time of task t_i if it is scheduled into VM v_j. This means that v_j accepts all the dependent data of t_i and and the task actually starts executing.

$$AST(i, m) = \begin{cases} \max\limits_{\substack{t_j \in {}^*t_i \\ n \neq m}} \{AFT(j, n) + x_{j_i}, AFT(j, m), AIT(m)\} & \text{if } {}^*t_i \neq \emptyset \\ AIT(m) & \text{if } {}^*t_i = \emptyset \end{cases} \tag{12}$$

$EFT(i, j)$ indicates the expected finishing time of task t_i if t_i is scheduled into VM v_j. It is estimate by task's starting time and the performance of virtual machines.

$$EFT(i, j) = \begin{cases} AST(i, j) + EET(i, j, AST(i, j)) \\ \qquad \text{if } t_i \text{ is in execution} \\ EST(i, j) + EET(i, j, EST(i, j)) \\ \qquad \text{if } t_i \text{ is waiting for execution} \end{cases} \tag{13}$$

$EIT(j)$ is the estimated idle time of the virtual machine, indicating that all tasks on the virtual machine have been completed.

$$EIT(j) = \max\{EFT(i, j)\} \tag{14}$$

$ET_{cp}(i, m, \theta)$ is the estimated duration of the critical path from t_i if it's scheduled into VM v_m on condition that its execution begins at time θ. Its value is calculated by the estimated execution time.

$$ET_{cp}(i, m, EST(i, m)) = EET(i, m, EST(i, m)) + \max_{t_j \in t_i^*}\{ET_{cp}(j, m, EST(j, m))\}\ if\ t_i^* \neq \emptyset \tag{15}$$

$$ET_{cp}(i, m, EST(i, m)) = EET(i, m, EST(i, m))\ if\ t_i^* = \emptyset \tag{16}$$

$EET(i, j, \theta)$ denotes the expected duration of task t_i if it's scheduled into VM v_j on condition that its execution begins at time θ. As discussed earlier, performance of cloud infrastructures can be time-fluctuating. Thus, $f(\theta, k)$ denote the historical performance of k type virtual machine at time θ measured or obtained through system logles. $f^1(\theta, k)$ denotes the predicted future value of performanc.

$$\begin{cases} TaskSize(i) = \sum_{\theta=EST(i,j)}^{x} f^1(\theta, g(j)) \\ EET(i, j, EST(i, j)) = x - EST(i, j) \end{cases} \tag{17}$$

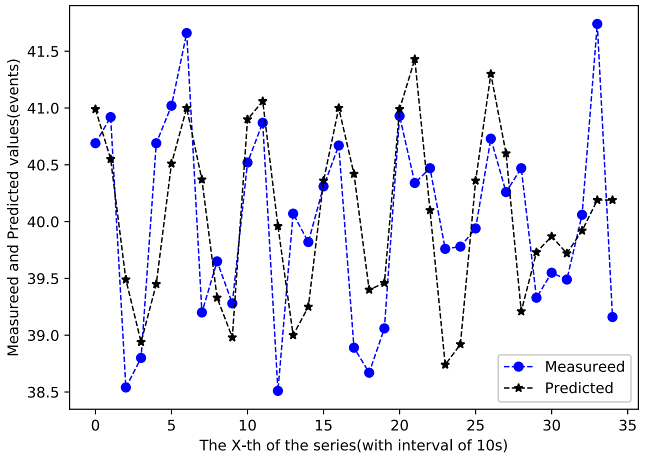


Fig. 5 Measured and predicted events of generateing prime numbers on Tencent cloud

6 Critical-path-duration-estimation-based VM selection

Algorithm 1 Scheduling($used\_VMs$, $taskQueue$, K).

```
1:  while $taskQueue \neq \emptyset$ do
2:      $t_i = taskQueue.deque$
3:      $cost_1 = INF$
4:      $cost_2 = INF$
5:      if $used\_VMs \neq \emptyset$ then
6:          $vm_1, cost_1 = cheapest\_UsedVMs(t_i, used\_VMs)$
7:          $vm_2, cost_2 = cheapest\_NewVM(t_i, K)$
8:      else
9:          $vm_2, cost_2 = cheapest\_NewVM(t_i, K)$
10:     end if
11:     //It is cheaper to use a VM has been used before
12:     if $cost_1 \leq cost_2$ and $cost_1 \neq INF$ then
13:         schedules.add($t_i$, $vm_1$)
14:     end if
15:     //It is cheaper to use a new virtual machine
16:     if $cost_2 \leq cost_1$ and $cost_2 \neq INF$ then
17:         $schedules.add(t_i, vm_2)$
18:     end if
19:     //The DeadLine is not expected to be met
20:     if $cost_1 = INF$ and $cost_2 = INF$ then
21:         $vm_3 = min\_time(K)$
22:         //Creating the most time-efficient virtual machine based on predicted performance
23:         if $^*t_i = \emptyset$ then
24:             Prompt the user to reset D
25:             Return $ET_{cp}(i, vm_3, timeNow)$
26:         else
27:             $schedules.add(t_i, vm_3)$
28:         end if
29:     end if
30: end while
```

As mentioned earlier, in order to yield dynamic workflow scheduling plans, we consider feeding predictive performance results into a Critical-Path-Duration-Estimation-based (CPDE for short) VM selection strategy. CPDE is implemented by applying the following algorithms.

Algorithm 1 This is the main function of the scheduling process. Parameter $taskQueue$ is the queue of tasks that can be currently scheduled. Parameter $used\_VMs$ is the set of VMs that has been used in scheduling algorithm. While a parent node of a task is running, if all of its parent have been scheduled, the child nodes are added to the schedulable queue, which is dynamically updated with execution. During the workflow execution, a task will adjust the scheduling scheme several times according to the execution situation. The algorithm is mainly to find the lowest cost case and generate scheduling plan and virtual machine creation plan.Parameter $schedules$ is the scheduling plan. When a task is being scheduled, there are three different possibilities:

- Using virtual machines in $used\_VMs$ can satisfies the deadline constraint. Find the cheapest scheduling scheme in this case and estimate the cost of the scheme based on the predicted performance. When this case happens, using a new virtual machine may also satisfy the deadline constraint, so need to compare the two cases.
- Createing a new irtual machine can satisfies the deadline constraint, it may happen whether $used\_VMs$ is empty or not. When this case happens, just using a previously used virtual machine may also satisfy the deadline constraint, so need to compare the two cases.
- The deadline constraint is not expected to be satisfied. If it is an task with no parents, prompt to reset D. Otherwise, create the least time-consuming virtual machine based on the predicted performance. Due to the deviation of actual performance, as the task continues, the predicted completion time may be possible to satisfy the deadline constraint again.

Algorithm 2 cheapest_UsedVMs(t_i, $used\_VMs$).

```
1:  $cost = INF$, $vm = NULL$
2:  for $v_j \in used\_VMs$ do
3:      $preStartTime = EST(i, j)$
4:      $taskExTime = EET(i, j, preStartTime)$
5:      //Estimated execution time of the task
6:      $cpExTime = ET_{cp}(i, j, preStartTime)$
7:      //Estimated execution time of the critical path
8:      $totalTime = cpExTime + preStartTime$
9:      $tempCost = taskExTime * h(v_j)$
10:     if $totalTime \leq D$ and $tempCost < cost$ then
11:         $cost = tempCost, vm = v_j$
12:     end if
13: end for
14: return $vm, cost$
```

Algorithms 2 This function is used to find the cheapest solution that satisfies deadline in the $used\_VMs$. Estimate the completion time of the task by Eq. 11 Eq. 14. The function evaluates Eqs. 15 and 16 to check if deadline is satisfied. Due to the fact that the scheduling scheme is generated dynamically and changes with time we calculate the estimated cost of the current task only.

Algorithm 3 cheapest_NewVM(t_i, K).

```
1: $cost$=$INF$, $vm$=$NULL$
2: $\theta = 0$
3: for $t_j \in {}^*t_i$ do
4:     if $\theta < EFT(j, w(j)) + x_{ji}$ then
5:         $\theta = EFT(j, w(j)) + x_{ji}$
6:     end if
7: end for
8: for $k \in K$ do
9:     $v_j$=newVM($k$)
10:    $//K$ is the set of VM types
11:    $preStartTime = \theta$
12:    $taskExTime = EET(i, j, preStartTime)$
13:    $cpExTime = ET_{cp}(i, j, preStartTime)$
14:    $totalTime = cpExTime + preStartTime$
15:    $tempCost = taskExTime * h(v_j)$
16:    if $totalTime \le D$ and $tempCost < cost$ then
17:        $cost = tempCost, vm = v_j$
18:    end if
19: end for
20: return $vm, cost$
```

Algorithms 3 Compute the case of scheduling a task to a newly created virtual machine. The estimated starting moment of the task deployed to a specific virtual machine is obtained according to Eq. 13 and the data transfer time. When calculating costs, we also only calculate the estimated cost of the current task, not the estimated cost of all tasks. A new virtual machine can be enabled directly at a specified time, without waiting for the used virtual machine to go idle.

Algorithm 4 Excution($schedules$).

```
1: while ${}^*t_i$ all finished do
2:     $t_i, v_j = schedules.get()//$Scheduling scheme
3:     if $v_j \notin used\_VMs$ then
4:         add $v_j$ to $used\_VMs$
5:     end if
6:     Waiting to execute $t_i$
7:     $update(AST(i, j))$
8:     $update(EFT(i, j))$
9:     $update(taskQueue)$
10:    $//$ When $t_i$ is running, update relevant data
11:    when the task $t_i$ is complete , $update(AFT(i, j))$
12: end while
```

Algorithms 4 This is pseudo-code when the schedule is executed. In the process of dynamic scheduling, the actual execution time of tasks can usually be different to that of predicted. The parent tasks may be completed earlier than expected, so tasks can be executed earlier and new virtual machine can be added earlier. As the task runs, much of the expected data can be dynamically updated. At the beginning of a task, its expected completion time is updated by the specific start time of the task. At this stage, the latest forecast data is updated and the list of tasks that can be scheduled is updated to calculate the new scheduling scheme.

Algorithm 5 $ET_{cp}(t_i, v_j, \theta)$.

```
1: $exTime = EET(i, j, \theta)$
2: $cpTime = exTime + \theta$
3: if $t_i{}^* \neq \emptyset$ then
4:     for $t_m \in t_i{}^*$ do
5:         $startTime = \theta + exTime$
6:         $temp = ET_{cp}(m, j, startTime)$
7:         if $temp > cpTime$ then
8:             $cpTime$=$temp$
9:         end if
10:    end for
11:    return $cpTime$
12: end if
13: if $t_i{}^* = \emptyset$ then
14:    // The task without child nodes
15:    return $cpTime$
16: end if
```

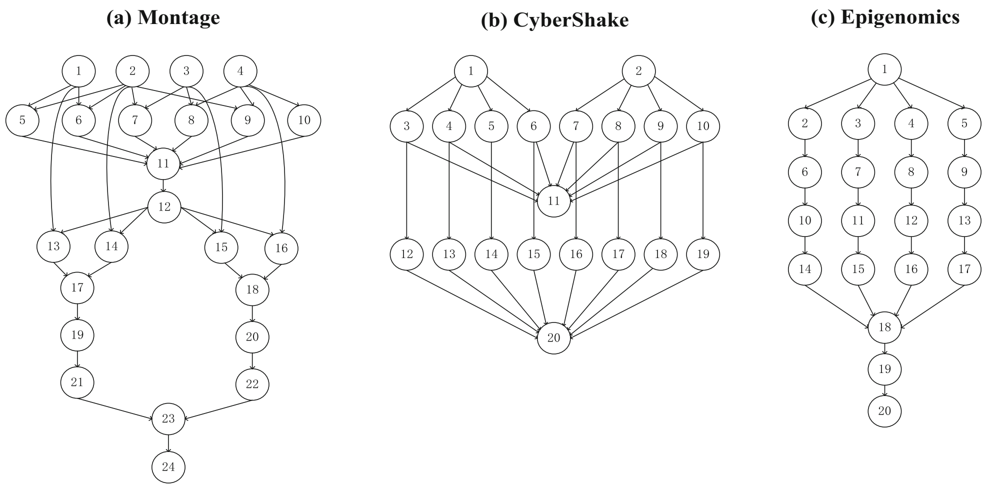
Algorithms 5 This pseudo-code illustrates the process of finding the critical path and calculating the estimated execution time. This function starts with t_i looking for the path with the longest execution time, until it finds the task without child task. We used the critical path execution time to estimate the completion time of the entire workflow. When multiple termination tasks occur, e.g., $t_j{}^* = \emptyset$ and $t_m{}^* = \emptyset$, this function is also applicable.

7 Experiments

We consider different classical scientific workflow templates, namely **Montage**, **CyberShake**, and **Epigenomics** as shown in Fig. 6 as the test cases. In these workflows, we consider that a workflow task corresponds to a procedure of repeatedly generating a given amount (4,000 times 100,000 or 2,000 times 100,000 in our case) of prime numbers. Define generating 100,000 primes as one event in the benchmark tool and the task sizes of the workflow are shown in Table 1.

We employ three industrial IaaS clouds, namely Baidu, Huawei, and Tencent to provide supporting VMs for the workflow tasks. VMs from Baidu, Huawei, and Tencent clouds are with different resource configurations, i.e., 1g RAM/1 core/50G storage for Baidu, 1g RAM/1 core/50G storage for Huawei, and 1g RAM/1 core/50G storage

Fig. 6 Overview of there workflow templates for the case study



for Tencent. The cost-per-second of these services are 0.34, 0.42, and 0.45 cents, according to their charging plans, respectively. The deadline constraints of Montage, CyberShake, and Epigenomics are 1010s, 340s, and 770s.

For the comparison reason, we consider PSO [36], GA [37], and MDGT [27] as the baseline algorithms. To validate the use of ARIMA-based prediction, we compare our method with a simplified version of our proposed method with pure CPDE but without ARIMA-based prediction.

As can be seen from Figs. 7, 8 and 9, our method clearly outperforms PSO, GA, and MDGT in terms of average cost for the three workflows (the averaged cost of our method for case1/case2/case3 are 825.4/740.3/680.4 cents, while those of GA for case1/case2/case3 are 826.1/741.1/681.9 cents, those of PSO for case1 /case2 /case3 are 825.7/740.8/681.5 cents, those of MDGT for case1 /case2 /case3 are 825.6/740.3/680.9 cents). Our method beats the pure CPDE-based one as well (the averaged cost of our method for case1/case2/case3 are 825.4/740.3/680.4 cents, while those of the pure CPDE-based one for case1/case2/case3 are 825.9/740.4/681.1 cents).

As can be seen from Figs. 10, 11 and 12 as well, our method clearly achieves lower average completion times by 0.84%, 1.1%, and 0.5% for three workflows, respectively.

Table 1 Scale of task

Task number	Event	Aount of prime numbers
$t_1, t_3, \ldots, t_{2n+1}$ $(2i+1 \leq n)$	4000	400M(4,000*100,000)
$t_2, t_4, \ldots, t_{2n}$ $(2i \leq n)$	2000	200M(2,000*100,000)

It's worth noting that such reduced completion times also lead to fewer violations to the deadline constraints. (the violation rates of our method for case1/case2/case3 are 0%/3%/0%, while those of GA for case1/case2/case3 are 40%/20%/13%, those of PSO for case1/case2/case3 are 30%/10%/20%, those of MDGT for case1/case2/case3 are 23%/20%/7%). Our method beats the pure CPDE-based one as well (the violation rates of our method for case1/case2/case3 are 0%/3%/0%, while those of the pure CPDE-based one for case1/case2/case3 are 3%/6%/3%).

Last but not the least, our proposed method has a lower time complexity than those of GA, PSO, and MDGT.The ARIMA model needs $O(\phi_1)$ time, where ϕ_1 denotes the bounded number of historical samples. The time to yield all predicted performance data for n tasks supported by m_1 types of VMs is $O(m_1 n \phi_1) = O(m_1 n)$. The time of function $ET_{cp}(\ )$, $Cheapest\_UsedVMs(\ )$, and $Cheapest\_NewVM(\ )$ are $O(n)$, $O(mn)$, and $O(m_1 n)$, where m indicates the number of VMs and m_1 denotes the number of virtual machine types. The time complexity for $Scheduling(\ )$ is thus $O((m+m_1)*n)$. Therefore, the time complexity of scheduling all tasks using our method is $O(n*(m+m_1)*n)+O(m_1 n \phi_1))=O((m+m_1)n^2)$.

The time complexity of GA is $O(ywn^2)$, where y denotes the size of initial population and w denotes the number of generation.The time complexity of PSO is $O(y_1 w_1 n^2))$, where y_1 denotes the number of particle clusters and w_1 denotes the number of iterations. The time complexity of MDGT is $O(kmn^2)$, where k denotes the number stages of the game. Note that y, w, y_1, and w_1 are usually large integers and thus GA, PSO, and MDGT can be much slower than our method.

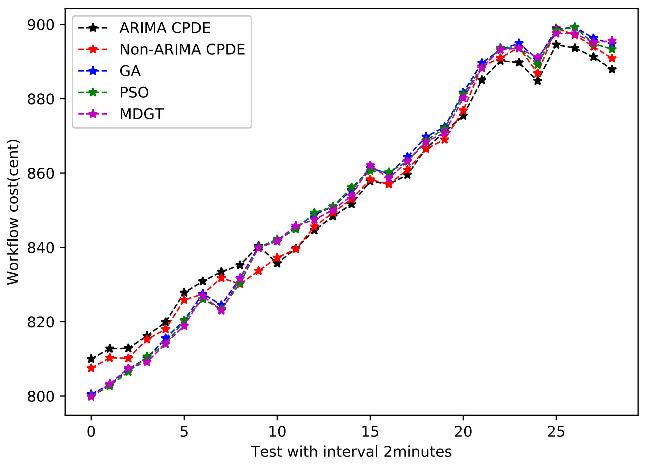


Fig. 7 Comparison of cost of Montage workflow

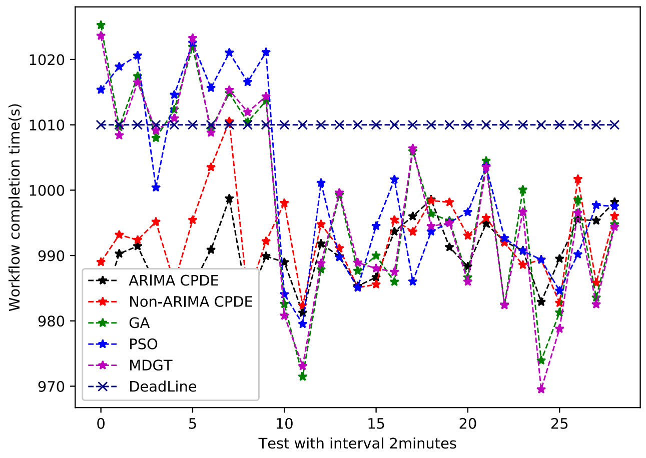


Fig. 10 Comparison of completion time of Montage workflow

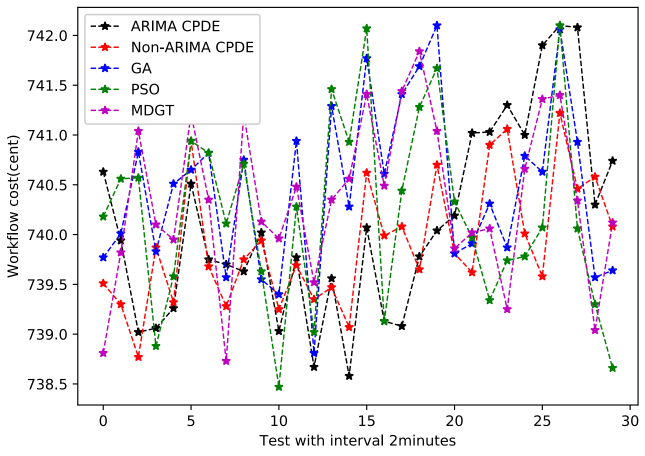


Fig. 8 Comparison of cost of CyberShake workflow

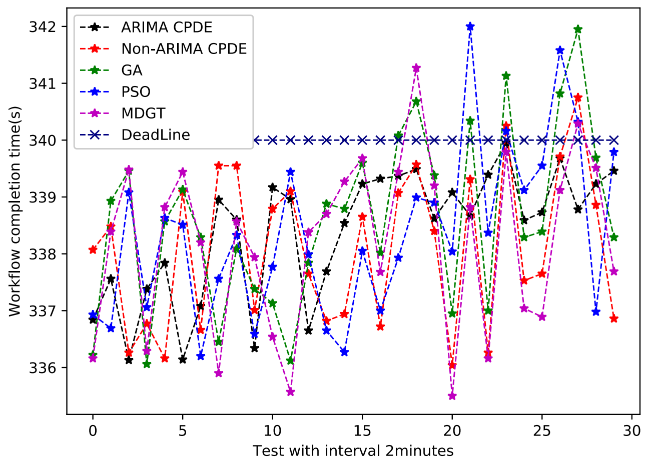


Fig. 11 Comparison of completion time of CyberShake workflow

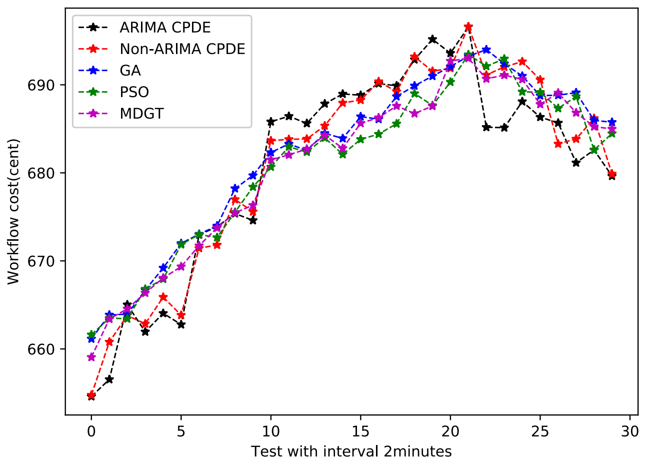


Fig. 9 Comparison of cost of Epigenomics workflow.

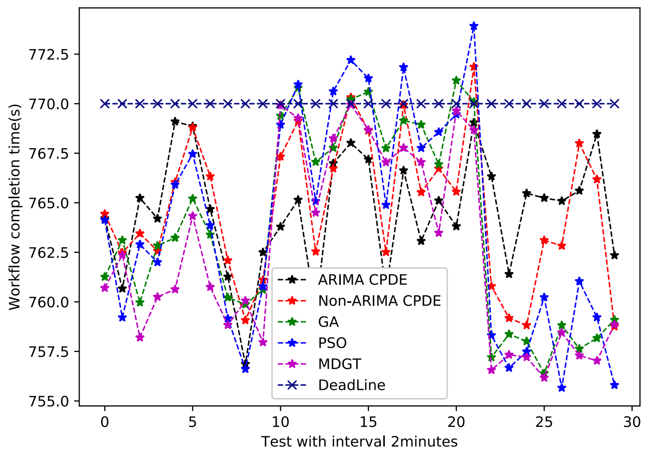


Fig. 12 Comparison of completion time of Epigenomics workflow

8 Conclusion

In this work, we develop a novel method for scheduling workflows upon IaaS clouds. Instead of assuming constant, stochastic, or bounded performance of VMs as most existing methods do, our proposed approach is capable of modeling time-varying performance and yielding cost-effective scheduling plans to reduce monetary cost while following constraints of Service-Level-Agreement (SLA). Our proposed approach leverages an ARIMA-time-series prediction model and a Critical-Path-Duration-Estimation-based VM Selection strategy which satisfies the constraint conditions according to the prediction results. A case study based on real-world third-party IaaS clouds and some well-known scientific workflow templates show that our proposed approach clearly outperforms traditional ones.

References

1. Xia Y, Zhou M, Luo X, Pang S, Zhu Q (2015) Stochastic modeling and performance analysis of migration-enabled and error-prone clouds. IEEE Transactions on Industrial Informatics 11(2):495–504
2. He Q, Han J, Yang Y, Jin H, Schneider J, Versteeg S (2014) Formulating cost-effective monitoring strategies for service-based systems. IEEE Trans Softw Eng 40(5):461–482
3. Yao Y, Cao J, Jiang Y, Wang J (2016) An optimal engine component placement strategy for cloud workflow service. In: IEEE international conference on web services, pp 380–387
4. He Q, Xie X, Chen F et al (2015) Spectrum-based runtime anomaly localisation in service-based systems. In: IEEE international conference on services computing, pp 90–97
5. Wu Q, Ishikawa F, Zhu Q, Xia Y, Wen J (2017) Deadline-constrained cost optimization approaches for workflow scheduling in clouds. IEEE Transactions on Parallel and Distributed Systems 28(12):3401–3412
6. He Q, Xie X, Wang Y et al (2017) Localizing runtime anomalies in service-oriented systems. IEEE Transactions on Services Computing 10(1):94–106
7. Xia Y, Zhou M, Luo X, Pang S, Zhu Q (2015) A stochastic approach to analysis of energy-aware DVS-enabled cloud datacenters. Systems Man and Cybernetics 45(1):73–83
8. Hwang S, Hsu C, Lee C (2015) Service selection for web services with probabilistic QoS. IEEE Trans Serv Comput 8(3):467–480
9. Xia Y, Zhou M, Luo X, Zhu Q, Li J, Huang Y (2015) Stochastic modeling and quality evaluation of infrastructure-as-a-service clouds. IEEE Trans Autom Sci Eng 12(1):162–170
10. Hadad JE, Manouvrier M, Rukoz M (2010) TQOs: transactional and QoS-aware selection algorithm for automatic web service composition. IEEE Trans Serv Comput 3(1):73–85
11. Wang Y, Liu H, Zheng W et al (2019) Multi-objective workflow scheduling with Deep-Q-Network-Based multi-agent reinforcement learning. IEEE Access, pp 39974–39982
12. Li W, Liao K, He Q, Xia Y (2019) Performance-aware cost-effective resource provisioning for future grid IoT-cloud system. Journal of Energy Engineering-asce 145(5)
13. Yin Y, Xia J, Li Y, Xu Y, Xu W, Yu L (2019) Group-wise itinerary planning in temporary mobile social network. IEEE Access, pp 83682–83693
14. Yin Y, Chen L, Xu Y, Wan J, Zhang H, Mai Z (2019) QoS prediction for service recommendation with deep feature learning in edge computing environment. Mobile Netw Appl. https://doi.org/10.1007/s11036-019-01241-7
15. Yin Y, Zhang W, Xu Y, Zhang H, Mai Z, Yu L (2019) Qos prediction for mobile edge service recommendation with auto-encoder. IEEE Access, pp 62312–62324
16. Gao H, Huang W, Duan Y, Yang X, Zou Q (2019) Research on cost-driven services composition in an uncertain environment. J Internet Technol 20(3):755–769
17. Gao H, Huang W, Yang X (2019) Applying probabilistic model checking to path planning in an intelligent transportation system using mobility trajectories and their statistical data. Intell Autom Soft Comput 25(3):547–559
18. Gao H, Miao H, Liu L, Kai J, Zhao K (2018) Automated quantitative verification for service-based system design: a visualization transform tool perspective. Int J Softw Eng Knowl Eng 28(10):1369–1397
19. Schad J, Dittrich J, Quianeruiz J (2010) Runtime measurements in the cloud: observing, analyzing, and reducing variance. Very Large Data Bases 3(1):460–471
20. Jackson K, Ramakrishnan L, Muriki K et al (2010) Performance analysis of high performance computing applications on the amazon web services cloud. In: IEEE international conference on cloud computing technology and science, pp 159–168
21. Wu Q, Zhu Q, Jian X, Ishikawa F (2014) Broker-based SLA-aware composite service provisioning. J Syst Softw, pp 194–201
22. Ibarra OH, Kim CE (1977) Heuristic algorithms for scheduling independent tasks on nonidentical processors. J ACM 24(2):280–289
23. Casas I, Taheri J, Ranjan R, Wang L, Zomaya AY (2016) GA-ETI: an enhanced genetic algorithm for the scheduling of scientific workflows in cloud environments. J of Comput Sci, pp 318–331
24. Verma A, Kaushal S (2017) A hybrid multi-objective Particle Swarm Optimization for scientific workflow scheduling. Parall Comput, pp 1–19
25. Zhou X, Zhang G, Sun J, Zhou J, Wei T, Hu S (2019) Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based HEFT. Futur Gener Comput Syst, pp 278–289
26. Wang Y, Jiang J, Xia Y, Wu Q, Luo X, Zhu Q (2018) A multi-stage dynamic game-theoretic approach for multi-workflow scheduling on heterogeneous virtual machines from multiple infrastructure-as-a-service clouds. In: International conference on services computing, pp 137–152
27. Sahni J, Vidyarthi DP (2018) A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment. IEEE Trans Cloud Comput 6(1):2–18
28. Mao M, Humphrey M (2011) Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In: IEEE international conference on high performance computing data and analytics
29. Calheiros RN, Buyya R (2014) Meeting deadlines of scientific workflows in public clouds with tasks replication. IEEE Trans Parall Distr Sys 25(7):1787–1796
30. Poola D, Garg SK, Buyya R, Yang Y, Ramamohanarao K (2014) Robust scheduling of scientific workflows with deadline and budget constraints in clouds. In: Advanced information networking and applications, pp 858–865

31. Ghosh R, Longo F, Frattini F, Russo S, Trivedi KS (2014) Scalable analytics for IaaS cloud availability. IEEE Int Conf Cloud Comput Technol Sci 2(1):57–70
32. Yin X, Ma X, Trivedi KS (2013) An interacting stochastic models approach for the performance evaluation of DSRC vehicular safety communication. IEEE Trans Comput 62(5):873–885
33. Zheng W, Zhou M, Wu L et al (2017) Percentile performance estimation of unreliable IaaS clouds and their cost-optimal capacity decision. IEEE Access, pp 2808–2818
34. Li W, Xia Y, Zhou M, Sun X, Zhu Q (2018) Fluctuation-aware and predictive workflow scheduling in cost-effective infrastructure-as-a-service clouds. IEEE Access, pp 61488–61502
35. Ma H, Zhu H, Hu Z, Tang W, Dong P (2017) Multi-valued collaborative QoS prediction for cloud service via time series analysis. Futur Gener Comput Syst 68(68):275–288
36. Coello CA, Pulido GT, Lechuga MS (2004) Handling multiple objectives with particle swarm optimization. IEEE Trans Evol Comput 8(3):256–279
37. Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197

Affiliations

Yi Pan[1] · Shu Wang[2] · Lei Wu[1] · Yunni Xia[3] · Wanbo Zheng[4] · Shanchen Pang[5] · Ziyang Zeng[6] · Peng Chen[7] · Yawen Li[3]

Yi Pan
cqupanyi@163.com

Shu Wang
shuwang@lnu.edu.cn

Yunni Xia
xiayunni@hotmail.com

Wanbo Zheng
zwanbo2001@163.com

Shanchen Pang
psc2008117@qq.com

Ziyang Zeng
57892653@qq.com

Peng Chen
chenpeng@mail.xhu.edu.cn

Yawen Li
yawenli@cqu.edu.cn

[1] School of Mathematical Science, University of Electronic Science and Technology of China, Chengdu, 611731, China

[2] College of Information, Liaoning University, Shenyang, 110036, China

[3] School of Computers, Chongqing University, Chongqing, 400044, China

[4] Data Science Research Center, Faculty of Science, Kunming University of Science and Technology, Kunming, 650500, China

[5] School of Computer and Communication Engineering, China University of Petroleum, Qingdao, 266580, China

[6] School of Mechanical Engineering, Chongqing University, Chongqing, 400044, China

[7] School of Computer and Software Engineering, Xihua University, Chengdu, 611731, China