IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 28, 2018, accepted September 3, 2018, date of publication September 13, 2018, date of current version November 9, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2869827

Fluctuation-Aware and Predictive Workflow Scheduling in Cost-Effective Infrastructure-as-a-Service Clouds

WEILING LI[1,2], YUNNI XIA[1,2], (Senior Member, IEEE), MENGCHU ZHOU[3,4], (Fellow, IEEE), XIAONING SUN[1,2], AND QINGSHENG ZHU[1,2]

[1]Software Theory and Technology Chongqing Key Lab, Chongqing University, Chongqing 400030, China
[2]Key Laboratory of Dependable Service Computing in Cyber Physical Society, Ministry of Education, Chongqing University, Chongqing 400030, China
[3]Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102, USA
[4]Institute of Systems Engineering, Macau University of Science and Technology, Macau 999078, China

Corresponding author: Yunni Xia (xiayunni@hotmail.com)

This work was supported in part by the International Joint Project through the Royal Society of the U.K., in part by the National Natural Science Foundation of China under Grant 61611130209, in part by the National Science Foundations of China under Grants 61472051/61702060, in part by the Science Foundation of Chongqing under Grant cstc2017jcyjA1276, in part by the China Postdoctoral Science Foundation under Grant 2015M570770, in part by the Chongqing Postdoctoral Science special Foundation under Grant Xm2015078, in part by the Universities' Sci-tech Achievements Transformation Project of Chongqing under Grant KJZH17104, in part by FDCT (Fundo para o Desenvolvimento das Ciencias e da Tecnologia) under Grant 119/2014/A3, and in part by the Chongqing grand RD Projects cstc2017zdcy-zdyf0120 and cstc2017rgzn-zdyf0118.

ABSTRACT Cloud computing is becoming an increasingly popular platform for the execution of scientific applications such as scientific workflows. In contrast to grids and other traditional high-performance computing systems, clouds provide a customizable infrastructure where scientific workflows can provision desired resources ahead of the execution and set up a required software environment on virtual machines (VMs). Nevertheless, various challenges, especially its quality-of-service prediction and optimal scheduling, are yet to be addressed. Existing studies mainly consider workflow tasks to be executed with VMs having time-invariant, stochastic, or bounded performance and focus on minimizing workflow execution time or execution cost while meeting the quality-of-service requirements. This work considers time-varying performance and aims at minimizing the execution cost of workflow deployed on Infrastructure-as-a-Service clouds while satisfying Service-Level-Agreements with users. We employ time-series-based approaches to capture dynamic performance fluctuations, feed a genetic algorithm with predicted performance of VMs, and generate schedules at run-time. A case study based on real-world third-party IaaS clouds and some well-known scientific workflows show that our proposed approach outperforms traditional approaches, especially those considering time-invariant or bounded performance only.

INDEX TERMS IaaS cloud, workflow, service-level-agreement, scheduling, quality-of-service (QoS).

LIST OF ABBREVIATIONS

DAG	Directed-Acyclic-Graph
GA	Genetic algorithm
HPC	High-performance computing
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
PM	Physical machine
SaaS	Software as a Service
SLA	Service-level-agreement
VM	Virtual machine

LIST OF SYMBOLS

α_i	The start time of an available period of VM v_i
β_i	The end time of an available period of VM v_i
γ_i	The estimated time that all tasks deployed on the same VM earlier than t_i must take
δ	The estimated start time for a scientific workflow
τ	Estimated workflow completion time
ω	The number of generations
ψ	The number of historical samples
A	The set of available VMs

b_i	The estimated start time of t_i
K	The set of VM types
C	Workflow cost
d_i	The estimated end time of t_i
D	The user-requested workflow completion time
$e_{i,j}$	The edge connecting t_i and t_j
f_i^j	The predicted future value of s_i^j
$g(j)$	The function to identify the type of v_j
$h(k)$	The function to identify cost-per-unit-time of using a type r_k VM
$l(i)$	The function to identify the index of t_i
m	The number of tasks
n	The number of types of VMs
N^+	The set of positive integers
r_i	The i^{th} type
R^+	The set of positive real numbers
s_i^j	The historical series of the execution time of t_i on any VM with type r_j
t_i	The i^{th} task of a scientific workflow
${}^\bullet t_i$	The parent sets of t_i
$t_i^\bullet$	The child set of t_i
T	The set of tasks of a scientific workflow
u_i	The earliest possible time to execute t_i
v_i	The i^{th} VM
$w(i)$	The function to identify the VM to which task t_i is to be scheduled
$x_{i,k}$	The transfer time between t_i and t_k
y	The size of initial population

I. INTRODUCTION

Workflows [1] are frequently employed to orchestrate data and computation-intensive scientific and engineering tasks in large-scale scientific applications, e.g., high energy physics and molecular biology. Scientific workflows aim at integrating data and computing steps into configurable, structured processes that perform semi-automated computational tasks for scientific applications. They usually present graphical interfaces to combine different technologies along with efficient methods for using them, and thus increase the efficiency of scientists. They are usually represented as directed acyclic graphs (DAGs) with their nodes representing discrete computational components and the edges representing connections along which data and results can communicate among components. Their capacity varies with the type of scientific applications. Their execution needs computing platforms with high performance, e.g., cluster and grid.

Recently, cloud computing is recognized as a promising solution and paradigm for providing a flexible, on-demand computing infrastructure over the Internet for large-scale scientific applications [2]. In a cloud computing system, physical and virtual resources can be allotted to combinations of one or more groups of users, with the owners of the resources deciding when and to whom they should be allotted. In this manner, collaborations can integrate pools of cloud resources to give supercomputer-class capability for large-scale scientific application to their users. A cloud management process allows end or tenant users to secure and release computing resource through a pay-as-you-go manner. The scientific applications can therefore elastically scale their resource pools upward or downward at run-time. A cloud management process only allocates required or estimated computing resources to achieve high utilization rate of resources and reduce operational cost. Cloud users are therefore charged using a pay-per-use price model based on the number of resources actually consumed.

Cloud computing systems are based on sharing of resources to achieve coherence and economies of scale, similar to utility (like electricity grid) on a network. Through the provision of on-demand access to computational resources, they offer services at three different levels: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS). IaaS clouds offer users with resources in the form of virtual machine (VM) instances deployed in a provider's data center. PaaS clouds offer platforms for users to design and implement their applications. SaaS clouds offer web applications/software over the Internet, running on cloud infrastructure. PaaS and SaaS clouds are thus less suitable for scientific applications than IaaS ones because they offer merely an environment to design, develop and test web based applications. Scientific workflows can be deployed and scheduled on IaaS clouds through two steps [3]: 1) a bag of physical resources are selected from the resource pool to run scientific tasks; and 2) a schedule is produced and then the corresponding task-resource mapping is performed.

Recently, the performance issues of scientific-workflow-oriented clouds and their scheduling attract considerable research attentions [27], [28]. A major difficulty in guaranteeing user-perceived performance of IaaS clouds lies in that VMs are subject to unexpected performance fluctuations. Schad *et al.* [4] show that the performance of VMs in Amazon EC2 cloud can vary by 24% under high workload. Jakson *et al.* [5] observe that performance variation of VMs can be as high as $30 - 65\%$ when data transfer among cloud nodes is unstable. Such fluctuations and variations of VMs could potentially impact the overall user-perceived quality of cloud systems, especially when the SLA thresholds [29], e.g., workflow execution time, are breached. Note that performance fluctuations of VMs could lead to increased operational cost as well since more PMs need to be turned on and VMs invoked if VMs already in use fail to accomplish their tasks.

It is therefore clear to see that run-time performance fluctuations of VMs significantly impact the scheduling of scientific workflows deployed on IaaS clouds. Instead of considering constant, stochastic (with derived or assumed probabilistic distributions), or bounded performance of VMs by most existing works discussed in the next section, we take their run-time performance fluctuations into account and

employ a time-series-based model to capture their run-time trends and predict their future performance. We then feed the predicted performance values into a genetic algorithm (GA) to deploy each task to an appropriate resource, aiming at minimizing the cost of running workflow while violating no SLA. The proposed framework captures other characteristics of IaaS cloud provisioning, e.g., on-demand resource encapsulation, elasticity, and pay-per-use pricing as well. To validate our proposed framework, we test our proposed algorithm to schedule some well-known scientific workflows deployed on real-world third-party IaaS clouds, namely Huawei, Amazon EC2, and Tencent clouds. It is observed that our method achieves lower averaged workflow completion time and workflow cost than traditional approaches at run-time. It is worth noting that our approach achieves lower SLA violation rates as well.

II. RELATED STUDIES

It is widely acknowledged that to schedule multi-task workflow on distributed platforms is an NP-hard problem [6]. It is therefore extremely time-consuming to yield optimal schedules through traversal-based algorithms. Fortunately, heuristic and meta-heuristic algorithms with polynomial complexity are able to produce approximate or near optimal solutions of schedules for Grid, Cluster, and cloud computing at the cost of some optimality loss.

For instance, Mao and Humphrey [10] develop a Scaling-Consolidation-Scheduling algorithm to schedule workflows on cloud. Their algorithm aims at finding optimal schedules to consolidate heterogeneous VMs. They consider a constant amount (20%) of performance variation. Meena *et al.* [7] consider a similar bounded performance variation and use a genetic algorithm to generate schedules. Malawski *et al.* [11] introduce three algorithms, DPDS (Dynamic Provisioning Dynamic Scheduling), WA-DPDS (Workflow-Aware DPDS), and SPSS (Static Provisioning Static Scheduling), to run multiple workflows in clouds. They aim at maximizing the number of workflows executed under given constraints of deadline and cost. However, they consider workflow tasks to be of constant execution time when executed on VMs.

The studies [12]–[14] propose heuristic algorithms for scheduling a single workflow instance on IaaS clouds. Abrishami *et al.* [12] introduce a static-Partial-Critical-Path procedure to evaluate the latest completion durations, then search through each partial critical path, and finally associate tasks on the partial critical path with the most inexpensive VM instances. If the algorithm fails to identify any available VM following the constraint of completion deadlines, it generates a new cheapest VM instance that executes all the tasks before its latest completion time. Calheiros and Buyya [13] consider soft deadlines, i.e., SLA violation rate. They propose a partial path identification algorithm that leverages idle periods of provisioned resources to improve the performance. They assume bounded (up to 10%) performance variation, in terms of execution time, of VMs. Poola *et al.* [14] present a similar framework and consider faulty-tolerance in finding optimal schedules.

Byun *et al.* [15] introduce a Balanced-Time-Scheduling (BST) algorithm for grid-based workflows to calculate minimally required numbers of physical resources to fulfill a completion-time constraint. BTS delays a task as much as possible on condition that its time constraint is not violated. However, they consider homogeneous VMs for simplicity. Later, Byun *et al.* [16] propose an improved Partitioned-Balanced-Time-Scheduling (PBTS) algorithm for cloud-based workflow scheduling. It evaluates the minimum capacity of resources required to execute a workflow by its given deadline. It assumes homogeneous VMs.

Wu *et al.* [17] propose an execution-time-reduction procedure for completion-time-constrainted workflows. They assume lower and upper bounds for the number of VMs required to meet the task deadlines. Then they develop a heuristic algorithm to deploy tasks to the allocated VM instances and employ an hour-minimization procedure to minimize the instance time taken by VMs.

Another category of solutions are based on meta-heuristic algorithms. For instance, Pandey *et al.* [18] aim at minimizing the operational cost of a single workflow while balancing the load on the available resources. They consider a fixed bag of VMs in the resource pool to support workflow tasks. Rodriguez *et al.* [19] propose a Particle-Swarm-Optimization (PSO) scheduling algorithm. It encodes particles based on the index of the resources that stand for the position of a particle. Nevertheless, it is stipulated that particles keep moving in different dimensions and thus the overall optimality of solutions is not guaranteed. Chen *et al.* [20] employ a similar encoding scheme and introduce a completion-time-constraint strategy for cost reduction based on the dynamic objectives. They consider a dynamic objective formulation which aims at time reduction instead of cost reduction when no feasible solution exists. Zhu *et al.* [8] present a similar optimization formulation but they consider only VMs with invariant and constant performance.

It can be seen that a major limitation of existing work is that constant or bounded performance of VMs is assumed. The limitation is multi-fold: 1) real-world clouds, especially heterogeneous and distributed cloud data-centers for scientific computing applications, are usually subject to performance and quality fluctuations at run-time. Such fluctuations are caused by, e.g., deteriorating/recovering network connectivity among cloud nodes and dynamic speed scaling of machines. Assuming constant VM performance, usually calculated as averaged historical performance, and using them as algorithm inputs enable a scheduling algorithm to produce fixed schedules that ignore the dynamic changes of system capability. Such schedules may lead to high SLA violation rates and bad user-perceived quality especially when clouds are under high stress; 2) employing bounded performance of VMs as algorithm inputs intends to avoid high SLA violation rates. However, such assumption can lead to the pessimistic estimation of system capability

and resource waste. Consider a cheap VM with fluctuating performance and with averaged/highest execution time of 10s/13s and another expensive VM with averaged/highest execution time of 7s/8s. If cloud users tolerate no more than 12s, the scheduling algorithm taking bounded performance as inputs probably choose the expensive one to avoid SLA violation. However, 13s happens only when the cloud is under high stress and a smarter algorithm is supposed to decide the trend (up or down) of performance change, predict the future performance of VMs, and choose from candidate VMs accordingly; 3) the work [21], [22] considers VM execution time to follow an exponential distribution. Although such assumption leads to Markovian models that are easy to solve, it is in practice, however, unrealistic in a real system because it implies future behaviors do not depend on the past history but the current status only; 4) some recent work [23] assumes general distributions instead of exponential ones and employs a Pareto distribution as a corresponding approximation type. They consider the historical empirical distribution of task processing time to be a right distribution to describe its future distribution. Similarly, Dong *et al.* [24] consider a novel mechanism that estimates the probability distribution of subtask execution time based on background VM load. It also introduces an elastic performance stochastic scheduling algorithm based on the derived stochastic distribution. A major limitation of random-distribution-based approaches lies in that the historical empirical distribution merely employs the density of samples as their distributional probabilities but ignores its trend and runtime fluctuations. As a case discussed in our earlier work [26], a distribution of task execution time with increasing occurrences of long delays with time, e.g., a sample distribution of ($d = 1$ *when* $t = 1s$, $d = 2$ *when* $t = 2s$, *and* $d = 3$ *when* $t = 3s$), clearly suggests a deteriorating performance. However, its empirical distribution may be quantitatively identical to that of another response delay type with the opposite behavior, i.e., $d = 3$ *when* $t = 1s$, $d = 2$ *when* $t = 2s$, *and* $d = 1$ *when* $t = 3s$. The above limitations could be well avoided by using a time-series-based analysis and prediction method instead. We, therefore, introduce a dynamic prediction approach by using the Autoregressive-Moving-Average-Model (ARIMA) series model [25] with special attention to the run-time trend of performance of VMs. We then feed a genetic algorithm predicted input performance of individual VMs, and generate schedules at run-time.

III. SYSTEM MODEL

A scientific workflow is described by a Directed-Acyclic-Graph (DAG) $W = (T, E)$ where $T = (t_1, t_2, ...t_m)$ denotes the set of tasks and E the set of edges. Without loss of generality, t_1 and t_m are considered to be the entry and exit tasks (note that a dummy entry/exist task with zero execution time can be added), respectively. The edge $e_{i,k}$ indicates that t_k can be executed after t_i is accomplished. ${}^\bullet t_i$ and $t_i^\bullet$ denote the parent and child set of t_i, respectively. The workflow starts and concludes by executing the entry and exit tasks, respectively. D denotes the user-recommended constraint of the completion time of the workflow, usually expressed in SLA documents. Note that this constraint can be either hard or soft one. In this work we consider hard one where the actual workflow completion time is bounded by D. A sample workflow is illustrated in Fig. 10.

An IaaS cloud supports scientific workflows through VMs. These VMs are selected from a VM pool, $A = \{v_1, v_2, ...v_m\}$. At most m VMs are required at runtime if no two tasks share the same VM. VMs can be different in their CPU speed, memory, and pricing configurations. We use $K = \{r_1, r_2, ...r_n\}$ to denote the set of VM types and a function, $g : A \rightarrow K$, to identify the type of each VM in the pool. Note that the mapping of VMs to their types can be dynamically determined at runtime for a performance/energy tradeoff purpose. VMs are charged based on their types. We therefore employ a function, $h : K \rightarrow R^+$, to identify the cost, in terms of dollars per unit time, of each type. Each VM has an available period for tasks. We use α_i and β_i to denote the start and end time of an available period of v_i. It is assumed that a VM can execute only one task at a time.

The execution order of a workflow can be expressed by assigning an index to each task. The index ranges from 1 to m and the ith item indicates the order of executing t_i. The relationship between each task and its index can be described by a function $l : T \rightarrow N^+$ and encoded as a vector containing a permutation of 1 to m. If i occurs before k in order, it does not necessarily indicate that the execution of t_i is earlier than t_k unless they are deployed on the same VM. The start time of tasks is decided by their supporting VMs and the completion time of their preceding tasks.

If task t_i connects t_k through edge $e_{i,k}$ and they are executed by different VMs, the transfer time, $x_{i,k}$, is inevitable because inter-VM data and control signal transfer is required. Otherwise, $x_{i,k} = 0$ if both tasks are on the same VM.

Workflow tasks executed by different types of VMs usually exhibit varying performance. Moreover, a task executed by the same VM at different time exhibits fluctuating performance as mentioned earlier. In order to capture the trend of performance variations at run-time and decide a schedule according to predicted future performance, we need to know the historical execution time, in terms of time series, of each task on each VM.

IV. ARIMA MODEL

Time series is a series of observations over one object or phenomenon based on time. It is widely used in economics, business, engineering, natural sciences, and social sciences. The salient feature of the time series is the serial dependency, i.e., the correlation of adjacent observations. The basic idea of the time-series-based prediction method is that the historical data of the time series reveals the changing trend with time, and extends the law to the future so as to predict the future values. In this paper, we consider an ARIMA model [25] as the prediction model of time-varying QoS of web services.

A time series is considered to be stationary only if its residuals are statistically independent of each other and constant in mean and variance over time. An ARIMA model is a non-stationary time series that can be modeled by an ARIMA model on condition that it can be converted into a stationary time series through differentiation.

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

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

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

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

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) model and a moving-average model:

$$\phi(B)x_t = \theta(B)z_t, \quad (3)$$

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

$$\phi(B) = (1 - \phi_1 B - \phi_2 B^2 - \cdots - \phi_p B^p). \quad (4)$$

$$\theta(B) = (1 + \theta_1 B + \theta_2 B^2 + \cdots + \theta_q B^q). \quad (5)$$

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

$$\varphi(B) = \phi(B)(1 - B)^d. \quad (6)$$

The predicted future values of an ARIMA series can thus be obtained as:

$$\varphi(B)x_t = \phi(B)(1 - B)^d x_t = \theta(B)z_t. \quad (7)$$

So

$$\phi(B)\omega_t = \theta(B)z_t, \quad (8)$$

where

$$\omega_t = (1 - B)^d x_t = \nabla^d x_t. \quad (9)$$

As for predictive service composition, we consider the ARIMA model described in this section as the underlying prediction method to process historical QoS data and obtain predictive QoS values. Such predictive values of candidate atomic services are fed into genetic algorithms to generate service composition plans.

As discussed earlier, the time required for a VM to execute a workflow task can be time-varying. We use s_i^j to denote a historical series, measured or obtained through system log-files, of the execution time of task t_i on any VM with type r_j. We employ the predicted future value of the execution time, i.e., f_i^j, as the inputs of an evaluation model and the genetic algorithm presented later. As shown in Figs. 1-9, predicted execution times of different tasks (to be discussed and explained in the section of case study) on three types of VMs well converge to measured ones with high accuracy.

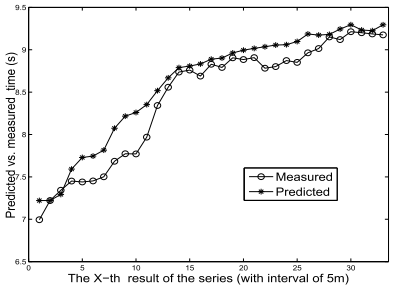


FIGURE 1. Measured vs. predicted execution time for the Gauss Legendre algorithm to compute 8 million digits of circumference ratio on Huawei cloud.

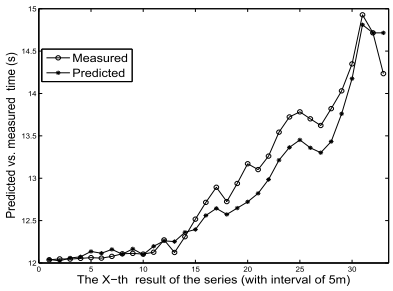


FIGURE 2. Measured vs. predicted execution time for the Gauss Legendre algorithm to compute 16 million digits of circumference ratio on Huawei cloud.

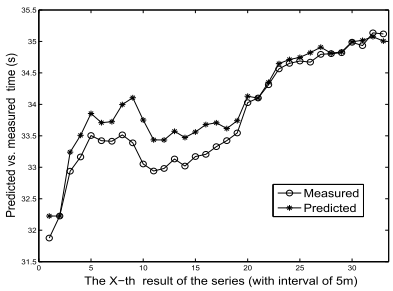


FIGURE 3. Measured vs. predicted execution time for the Gauss Legendre algorithm to compute 32 million digits of circumference ratio on Huawei cloud.

V. PROBLEM FORMULATION

High performance and low cost are usually contradicting goals when scheduling workflows in clouds. Thus, our proposed work tries to reconcile them and identify a cost-effective schedule to deploy scientific tasks on VMs in order to minimize the overall cost while satisfying SLA, thus guaranteeing that the workflow completion time is always bounded. The resulting problem can therefore be formulated

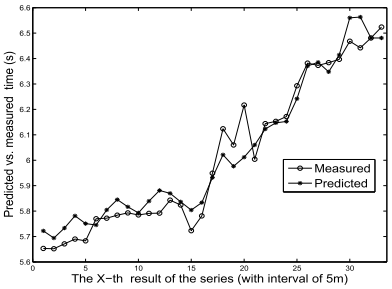


FIGURE 4. Measured vs. predicted execution time for the Gauss Legendre algorithm to compute 8 million digits of circumference ratio on Tencent cloud.

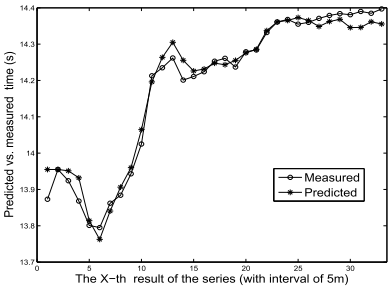


FIGURE 5. Measured vs. predicted execution time for the Gauss Legendre algorithm to compute 16 million digits of circumference ratio on Tencent cloud.

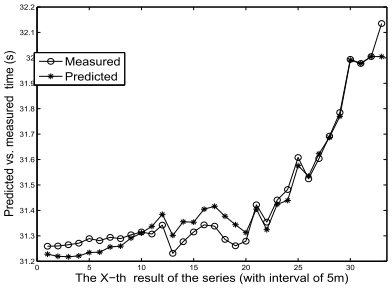


FIGURE 6. Measured vs. predicted execution time for the Gauss Legendre algorithm to compute 32 million digits of circumference ratio on Tencent cloud.

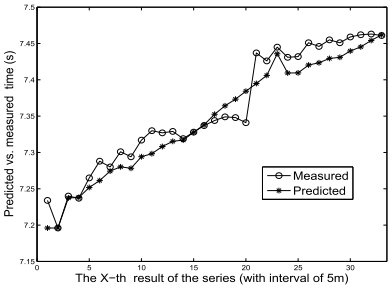


FIGURE 7. Measured vs. predicted execution time for the Gauss Legendre algorithm to compute 8 million digits of circumference ratio on Amazon cloud.

as:

$$\begin{aligned} &Min\ C \sum_{i=0}^{m} h(g(w(i))) \times f_i^{g(w(i))} \\ &s.t.\ \tau \leq D \end{aligned} \tag{10}$$

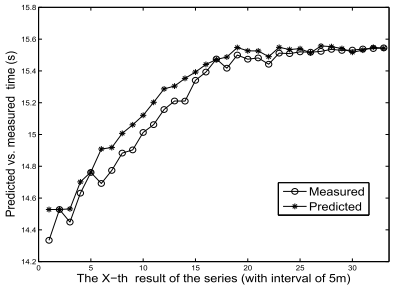


FIGURE 8. Measured vs. predicted execution time for the Gauss Legendre algorithm to compute 16 million digits of circumference ratio on Amazon cloud.

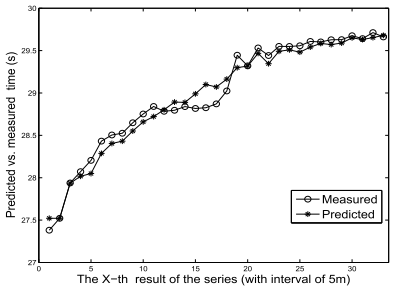


FIGURE 9. Measured vs. predicted execution time for the Gauss Legendre algorithm to compute 32 million digits of circumference ratio on Amazon cloud.

where C denotes the cost of running a scientific workflow, w the function of a schedule, $w(i)$ the VM to which task t_i is scheduled, $g(w(i))$ the type of VM to which task t_i is scheduled, τ the estimated time required to accomplish the workflow, $h(j)$ the function to identify cost-per-unit-time of using a type r_j VM, and $f_i^{g(w(i))}$ the predicted future value of the execution time of task t_i.

The derivation of τ requires some efforts. τ can be calculated as the estimated end time of the last task in the workflow:

$$\tau = d_m \tag{11}$$

where d_i denotes the estimated end time of task t_i.

d_i can be iteratively calculated as:

$$d_i = f_i^{g(w(i))} + b_i \tag{12}$$

where b_i denotes the estimated start time of executing t_i and $f_i^{g(w(i))}$ the predicted execution time of t_i itself.

b_i is decided by various factors, namely the available period of its supporting VM, the estimated end time of its immediately preceding tasks, and the time required for data transfer. Let γ_i denote the estimated time that all tasks deployed on the same VM earlier than t_i must take. We have:

$$\gamma_i = \max_j \{d_j | l(j) < l(i) \wedge w(i) = w(j)\} \tag{13}$$

where $l(j) < l(i)$ indicates that t_j's order index is smaller than that of t_i and $w(i) = w(j)$ means that t_i and t_j are scheduled into the same VM.

Note that the dependency constraint requires that a task be executed only if its all immediately preceding ones successfully terminate and transfer data. We use y_i to denote the estimated earliest time that the described condition holds for t_i.

$$y_i = max\{d_k + X_{k,i} | t_k \in^{\bullet} t_i\} \tag{14}$$

where ${}^{\bullet}t_i$ denotes the immediately preceding tasks of t_i, i.e., those which directly connect t_i through edges in the scientific workflow.

The earliest possible time to execute t_i can therefore be calculated as:

$$u_i = max\{\gamma_i, y_i\} \tag{15}$$

Based on the above observations, when $i > 1$, b_i can be obtained as:

$$b_i = \begin{cases} u_i & \text{if } u_i \geq \alpha(w(i)) \ \wedge u_i + \\ & f_i^{g(w(i))} \leq \beta(w(i)) \\ \infty & \text{else} \end{cases} \tag{16}$$

The above equation indicates that the earliest possible time to execute t_i should be later than the start time of the available period of its supporting VM and t_i should terminate before the end time of such period. Otherwise, the corresponding schedule is considered to be impossible and b_i is assigned ∞ accordingly.

The entry task of a scientific workflow has no preceding tasks and therefore its estimated end time is:

$$d_1 = \begin{cases} b_1 + f_1^{g(w(1))} & \text{if } b_1 + f_1^{g(w(1))} \leq \beta(w(1)) \\ \infty & \text{Otherwise} \end{cases} \tag{17}$$

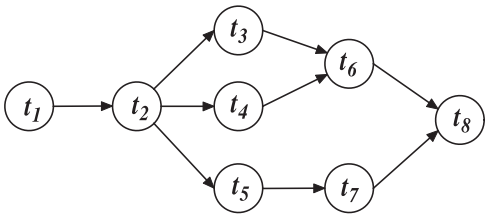
where b_1 can be obtained as:

$$b_1 = max\{\delta, \alpha(w(1))\} \tag{18}$$

where δ denotes the estimated start time for a scientific workflow's initialization and preprocessing.

VI. GENETIC ALGORITHM FOR WORKFLOW SCHEDULING

Since the resulting optimization problem is NP-hard, we have to rely on meta-heuristic algorithms in solving any sizable problem. Note that a significant number of studies, e.g., [7], [8], [20] clearly suggest the advantage of time-efficiency of genetic algorithms for workflow scheduling over other heuristics, e.g., Particle Swarm optimization and Ant Colony optimization. We therefore consider using GA empowered by time-series-based prediction and novel designs of genetic operations. GA falls into the class of evolutionary algorithms [36], [37]. It is a metaheuristic procedure similar to the process of natural selection. It is frequently used to yield high-quality solutions to optimization and searching problems by employing bio-inspired operations, e.g., mutation, crossover and selection. In it, a population of candidate solutions (called individuals) to an optimization problem keeps evolving toward better solutions. Every candidate solution is associated with multiple properties (called chromosomes or genotype) which can be mutated and altered; traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. Based on our problem descriptions, we present definitions of genetic operations next.



Encoding

Task-Index	1	2	5	4	3	7	6	8
Task-VM	2	2	1	1	3	4	2	1
VM-Type	1	1	2	3	4	4	4	4

FIGURE 10. A sample scientific workflow and its encoding scheme.

A. ENCODING

A schedule described by l, w, and g functions is expressed through chromosome described by 3 vectors of positive integers, namely *Task-Index*, *Task-VM* and *VM-Type*. The length of vectors is m, i.e., the number of tasks. *Task-Index* identifies the order of execution of each task by associating an index with each task. The initial order of tasks is based on the structural constraint and topological sort of a workflow itself, i.e., a task can never be executed before its immediately preceding ones.

The first step of encoding deals with *Task-Index* and makes a topological sort and then allocates an index number to every task according to the sorting levels. The index begins with 1, and t_i represents a task whose topological index is i. The second step deals with *Task-VM*. An index in it stands for a task and its value stands for the VM to which this task is scheduled. Similarly, the third vector *VM-Type* identifies the mapping from VMs to their types.

Fig. 10 also shows the encoding scheme for the sample workflow given earlier. In this schedule, each task is associated an execution order of [1, 2, 5, 4, 3, 7, 6, 8] through the *Task-Index* vector. The *Task-VM* vector suggests that only 4 VMs are used to support the workflow tasks. Note that t_3 and t_4 can be executed in parallel according to the structural constraint specified by its corresponding DAG even if t_4 is assigned with a higher index. However, t_3 and t_4 share v_1 and thus are actually sequentially executed.

B. CROSSOVER

A valid scheduling order is supposed to comply with the structural constraint of a scientific workflow, i.e., a task can never be executed before its preceding tasks. The crossover operation should comply with these restrictions. As shown in Fig. 11, the operator randomly selects a cutting point to split each parent vector, i.e., *Task-Index*, into two sub-vectors. Then, the two first sub-vectors are exchanged to

```
1 : Procedure CrossoverIndex(X, Y)
2 :   point←RandomInteger(1, m)
3 :   vec1←SubVector(X,1,point)
4 :   vec2←SubVector(Y,1,point)
5 :   for i←1 to m
8 :     if   X[i] not∈vec1
9 :          intert X[i] at the end of vec1
10:     endif
11:   endfor
12:   for i←1 to m
13:     if   Y[i] not∈vec2
14:          intert Y[i] at the end of vec2
15:     endif
16:   endfor
17:endprocedure
```

FIGURE 11. The crossover operation on execution order of workflow tasks.

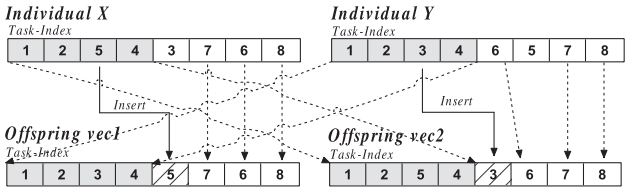


FIGURE 12. An example of a crossover operation on the order of workflow execution.

be the offspring, and the second sub-vectors are abandoned. In the following, each parent order vector is investigated from the beginning and any task that has not appeared in the first sub-vector is added to the end of this offspring. The crossover operator complies with dependency constraints since the orders of any two tasks already exist in no less than one parent. An illustration of the crossover applied to the workflow shown in Fig. 10 is in Fig. 12, where point 4 is randomly selected as the cutting point. The gray parts in both parent vectors are exchanged and the missing tasks caused by the swap operation, i.e., shaded ones in this figure, are inserted into the remaining vectors following their original orders. The time complexity of this operation is $O(m)$.

The crossover operation is also conducted on *Task-VM* and *VM-TYPE* vectors. Similarly, it randomly decides a cut-off point and swaps the first parts of two parent *Task-VM* vectors. However, the crossover operation itself may erase useful information when the swapped part changes the relationship between VMs and their corresponding types. Take Fig. 14 as an example. The cut-off point of 2 is randomly decided and the gray parts of two parents need to be

```
1: Procedure CrossoverVM(X, Y)
2:   point←RandomInteger(1, m)
3:   for i←1 to point
4:     v←X.Task-VM[i]
5:     tp1←X.VM-TYPE[v], tp2←Y.VM-TYPE[v]
6:     vinX= |{i|v= X.Task-VM[i] ∧ m=>i>point}|
7:     vinY= |{i|v= Y.Task-VM[i] ∧ m=>i>point}|
8:     if   tp1≠tp2
9:       if   vinX>vinY
10:        Y.VM-TYPE[v]←tp1
11:      else
12:        X.VM-TYPE[v]←tp2
13:      endif
14:    endif
18:    v←Y.Task-VM[i]
19:    tp1←Y.VM-TYPE[v], tp2←X.VM-TYPE[v]
20:    vinY= |{i|v= Y.Task-VM[i] ∧ m=>i>point}|
21:    vinX= |{i|v= X.Task-VM[i] ∧ m=>i>point}|
22:    if   tp1≠tp2
23:      if   vinY>vinX
24:        X.VM-TYPE[v]←tp1
25:      else
26:        Y.VM-TYPE[v]←tp2
27:      endif
28:    endif
29:    Swap(X.Task-VM[i], Y.Task-VM[i])
30:  endfor
31:  vec1←X, vec2←Y
32: endprocedure
```

FIGURE 13. The crossover operation on $Task-VM$ and $VM-TYPE$.

crossovered. We first consider t_1 of the first parent with its supporting VM v_2. v_2 has different corresponding types in the two parents, i.e., r_1 in X and r_4 in Y. Since v_2 appears more frequently, as shown in shaded parts, in X than in Y, such conflict can only be solved by reassigning the type of v_2 in Y as that of v_2 in X. Consequently, the type of v_2 in Y is changed into r_1. Similarly, we consider t_1 of Y. The supporting VM of t_1 in Y is v_3. v_3 has different corresponding types in the two parents, i.e., r_4 in Y and r_2 in X. Since v_3 appears less frequently in Y than in X, as shown in shaded parts, such conflict can only be solved by reassigning the type of v_3 in Y as that of v_3 in X. Consequently, the type of v_3 in Y is changed into r_2. After the change is made, a swap of the first items of two parents is conducted. In the next step, we consider t_2 of X and its supporting VM is v_2. v_2 has identical types in two parents and thus no conflict exists. Similarly, we then consider t_2 of Y. The supporting VM of t_2 in Y is v_4. v_4 has different types in two parents, i.e., r_2 in Y and r_3 in X. Since

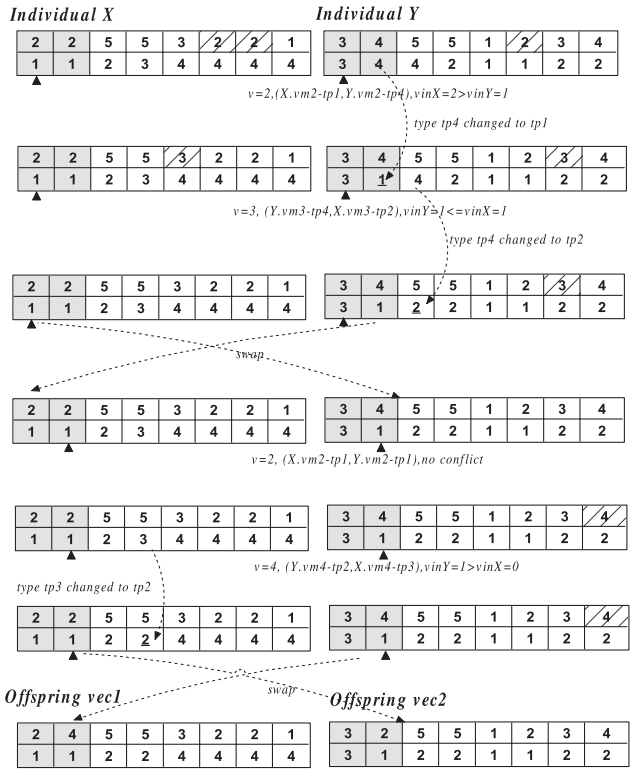


FIGURE 14. An example of crossover operations on *Task − VM* and *VM − Type* vectors.

v_3 appears more frequently in Y than in X, as shown in shaded parts, such conflict can only be solved by reassigning the type of v_4 in X as that of v_4 in Y. Consequently, the type of v_4 in Y is changed into r_2. After the change is made, a swap of the second items of two parents is conducted. Finally, X and Y are assigned to two offsprings. The pseudocode of the crossover operation is given in Fig. 13.

C. MUTATION

The mutation operation on the execution order of a workflow should comply with its structural constraint. Fig. 15 presents the pseudocode of this operation. The mutation operator randomly chooses a task t_i, randomly selects one of the tasks, which is not on the same path from the entry task to the exit task with t_i, and swaps the selected task with t_i. Since the mutation operations on *Task-VM* and *VM-Type* vectors have no worries of breaching the structural constraint of the workflow, they simply randomly generate a new feasible value for every position, with a small probability. Their pseudocode is therefore not shown.

D. INITIAL POPULATION

The search space of solutions usually grows exponentially with the scale of a scientific workflow. To accelerate the search speed and convergence of the genetic algorithm, the initial population are defined and generated in such a way that each individual of the population complies with the structural constraint of the scientific workflow and its

```
1 : Procedure Mutation(X)
2 :   point←RandomInteger(1, m)
3 :   count←0
4 :   for i←1 to m
8 :     if  t_X.Task-Index[i] and t_X.Task-Index[point] are not on
            the same path from t_1 to t_m
9 :         count ←count+1
10:    endif
11:  endfor
12:  if  count>0
13:      select ←RandomInteger(1,count)
14:      for i←1 to m
15:          if  t_X.Task-Index[i] and t_X.Task-Index[point] are not on
                 the same path from t_1 to t_m
16:              if  select=1
17:                  swap(t_X.Task-Index[i], t_X.Task-Index[point])
18:                  break
19:              else
19:                  select ←select-1
20:              endif
21:          endif
21:      endfor
22: endif
23:endprocedure
```

FIGURE 15. The mutation operation on *Task − Index*.

Task-Index, *Task-VM*, and *VM-Type* vectors are decided randomly. The pseudocode of generating an individual is given in Fig. 16. This procedure starts with deciding the execution order of the first/entry task and iteratively invokes an ancillary procedure, called *RandInit*(), to decide the execution orders of the remaining tasks. To decide the corresponding task to be executed at position i, *RandInit*() randomly selects one of the unexecuted tasks which is not on the path from any other unexecuted task to the exit task. When the execution orders of all tasks are decided, *Task-VM* and *VM-Type* vectors are randomly generated.

The fitness of an individual solution is decided by its minimization of estimated workflow cost on condition that SLA is satisfied. The selection strategy is therefore based on the fitness estimation and implemented through a tournament-based method [30]. To be specific, we employ a constraint handling strategy described below to select chromosomes for new generations: 1) if the τ values of two solutions are feasible, then the one with higher cost is discarded, 2) if the τ value of only one solution is feasible, then the infeasible one is simply discarded; and 3) if the τ values of both solutions are infeasible, then the one with higher estimated completion time is discarded.

```
1 : Procedure Initialization( )
2 :   RandInit(1)
3:   mark=[0,0,....0]
4: endprocedure

1 : Procedure RandInit(i)
2 :  if   i=1
3:       Task-Index[1] ←1
4:       mark[1] ←1
5:       RandInit(2)
6:  elseif   i<m
7:       j=RandChoose({k|Mark[k]=0, t_k is not on the path
         from any other unmarked task to the ending task})
8:      mark[j] ←1
9:      Task-Index[i] ←j
10:     RandInit(i+1)
11: else
12:      mark[m] ←1
13:      Task-Index[m] ←m
14:      for   x ←1 to m
15:           Taks-VM[x] ←RandInteger(1,m)
16:           VM-Type[x] ←RandInteger(1,p)
17:      endfor
18:      add Taks-Index, Task-VM, and VM-Type into the
         set of initial population
19: endif
20:endprocedure
```

FIGURE 16. The procedure to generate an individual into the initial population.

E. COMPLEXITY ANALYSIS

The overall computational complexity of our proposed framework can be analyzed by examining its initialization, selection, crossover, mutation and fitness evaluation operations. The time complexity of initializing an individual is $O(m+n)$, and thus population initialization requires $O(m+n) \times y$ where y is the size of initial population. The time complexity for selection, crossover, and mutation operations are $O(y)$, $O(m^2)$, and $O(m)$, respectively. Consequently, the total time complexity of selection, crossover, and mutation with ω generations is $O(\omega y) + O(\omega m^2) + O(\omega m)$. The fitness evaluation for each individual has the time complexity of $O(m^2)$ and thus fitness evaluation for initial population of size y with ω generations has the time complexity of $O(y\omega m^2)$. The total time complexity of selection, crossover, mutation and fitness evaluation is thus $O(m+n) \times y + O(y\omega m^2) + O(\omega y) + O(\omega m^2) + O(\omega m) = O(y\omega m^2)$. Note that the time complexity for the Box-Jenkins method is $O(\psi)$ where ψ is the number of historical samples to train an ARIMA model and ψ is usually bounded. The complexity for generating all predicted performance data for m tasks supported by n types of VMs is $O(mn\psi) = O(mn)$. The overall time complexity for our proposed framework is thus $O(mn) + O(y\omega m^2)$. Note that the number of types of VMs is usually smaller than the number of machines and thus the overall complexity can further be expressed as $O(y\omega m^2)$.

VII. CASE STUDY AND COMPARISON

In this section, we present a case study of real-world scientific workflows deployed on commercial IaaS clouds, to compare traditional scheduling approaches with our proposed framework. We employ three different classical scientific workflow templates, namely Montage, CyberShake, and Epigenomics, to support tasks of GaussLegendre calculations with a different number of digits. The GaussLegendre calculation is a highly-memory-requiring iterative procedure to compute the digits of circumference ratio to a specific number of digits. The procedure repeatedly replaces two numbers by their arithmetic and geometric mean, in order to approximate their arithmetic-geometric mean. This procedure is implemented by a benchmark tool, i.e., Super-Pi

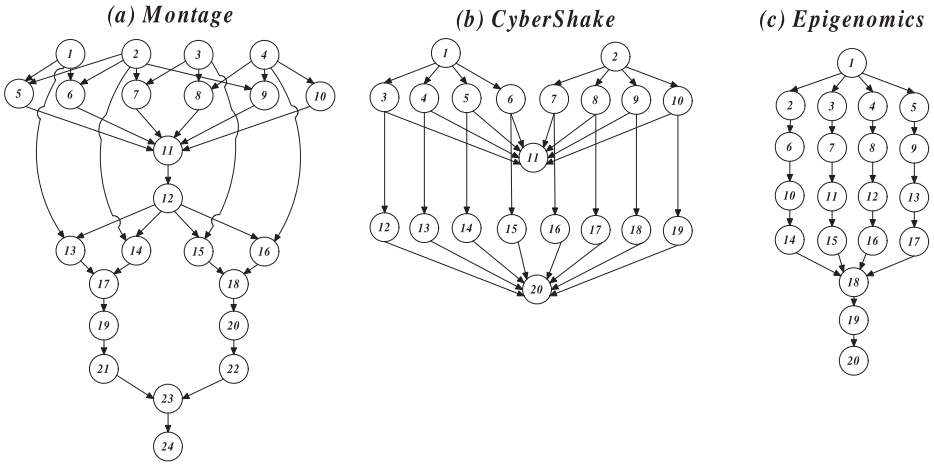


FIGURE 17. The scientific workflow templates for the case study.

TABLE 1. Tasks of three scientific workflows(million).

Montage	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
Digits of circumference ratio required	8	16	8	32	8	8	16	32	8	32
Montage	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}	t_{19}	t_{20}
Digits of circumference ratio required	8	16	32	32	8	16	16	8	32	8
Montage	t_{21}	t_{22}	t_{23}	t_{24}						
Digits of circumference ratio required	8	8	8	8						
CyberShake	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
Digits of circumference ratio required	8	8	8	8	8	32	16	8	32	32
CyberShake	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}	t_{19}	t_{20}
Digits of circumference ratio required	16	16	8	32	8	16	8	8	16	16
Epigenomics	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
Digits of circumference ratio required	32	32	16	16	16	16	8	8	8	8
Epigenomics	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}	t_{19}	t_{20}
Digits of circumference ratio required	8	8	8	8	8	8	8	8	8	8

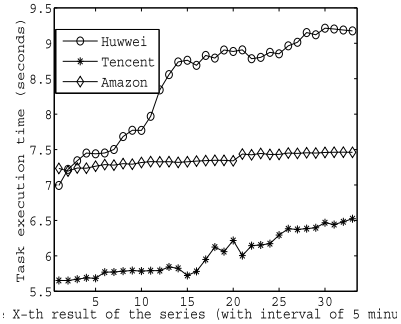


FIGURE 18. Measured time for the Gauss Legendre algorithm to calculate 8 million digits of circumference ratio.

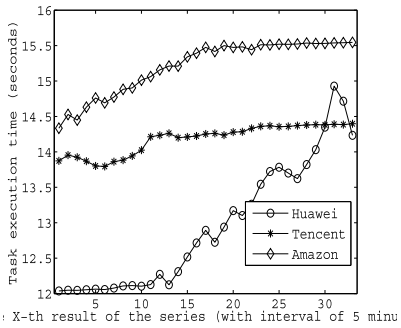


FIGURE 19. Measured time for the Gauss Legendre algorithm to calculate 16 million digits of circumference ratio.

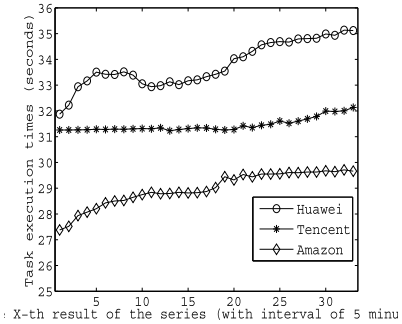


FIGURE 20. Measured time for the Gauss Legendre algorithm to calculate 32 million digits of circumference ratio.

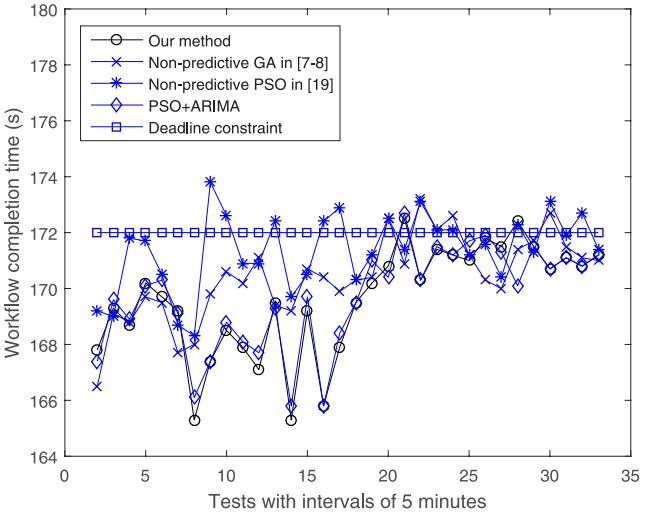


FIGURE 21. Comparison of completion time of Montage workflow.

(from http://www.superpi.net/). This tool is frequently used in testing floating-point performance of computing systems. Tasks of the sample workflows are required to run the Super-Pi tests with different requirements of the numbers of digits to generate as given in Table. 1. The maximum number of required VMs equals that of tasks and VMs are available from the beginning to the end. We consider 172s, 79s, and 110s as the bounds of completion time of three workflows.

We use three commercial IaaS clouds, namely Huawei, Tencent and Amazon EC2 to test the workflows and our proposed scheduling algorithm. Each commercial cloud provides one type of VM (1g RAM/1 core/40G storage for Huawei cloud, 1g RAM/1 core/60G storage for Tencent cloud, and 2g RAM/1 core/30G storage for Amazon EC2 cloud). Hence totally, we have three types of VMs available to support the workflow execution. It takes 0.06 seconds in average to transfer data between different clouds. The cost-per-second of these clouds are 1.5 cent, 1.6 cent, and 1.7 cent, respectively. The three deadlines are decided in a way that deadline constraints generally comply with the baseline performance of VMs used. As can be seen from Figs. 1-9, it generally takes 7, 13, and 30 seconds to execute three types of Gauss Legendre calculations on three types of VMs. Consequently, the baseline execution time for Montage, Cybershake, and Epigenomic workflows should fall into [150, 175], [70, 90], and [100, 120] based on their longest-path-distribution estimations in DAGs. The time unit is second. Consequently, the deadline should not be too low to avoid the case that

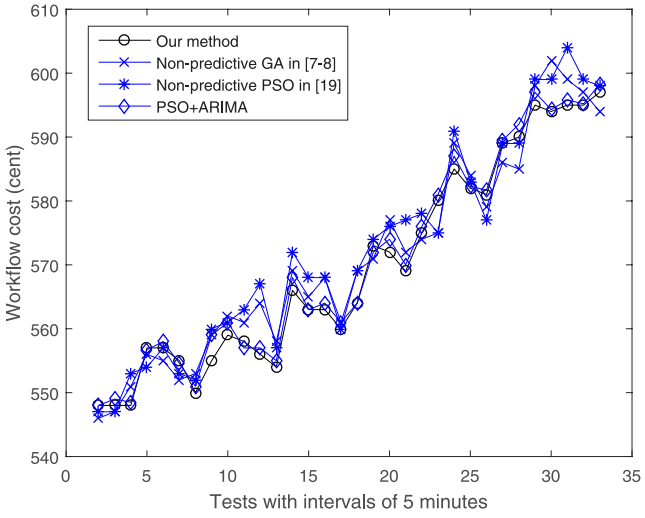


FIGURE 22. Comparison of cost of Montage workflow.

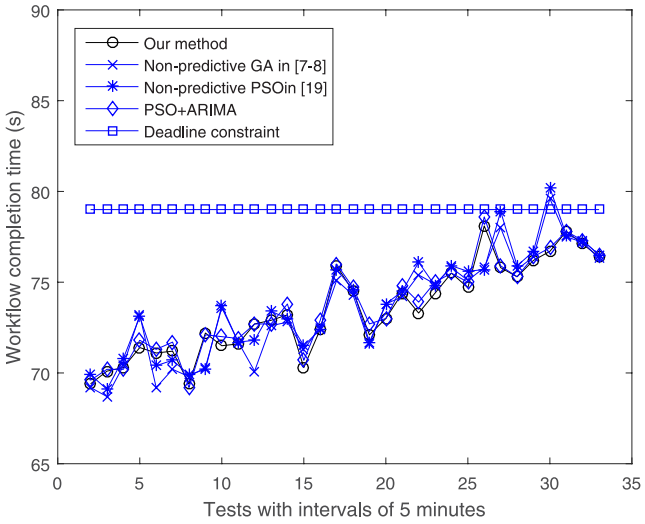


FIGURE 23. Comparison of completion time of Cybershake workflow.

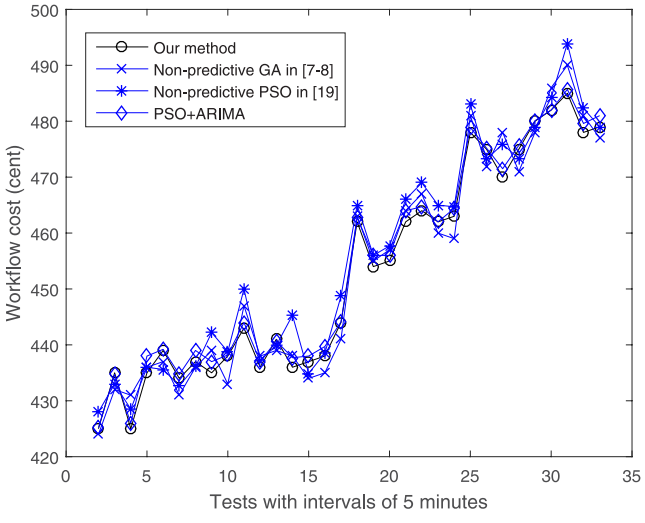


FIGURE 24. Comparison of cost of Cybershake workflow.

both our proposed method and traditional algorithms fail to work out. On the other hand, if the deadline is too high, all algorithms work very well and lead to no SLA violation.

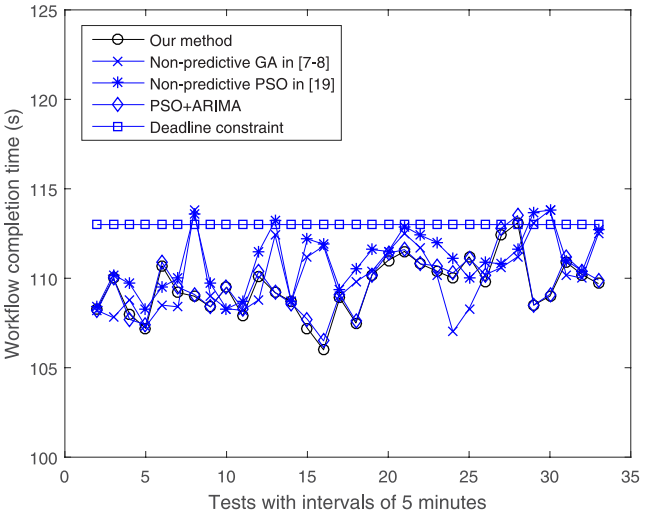


FIGURE 25. Comparison of completion time of Epigenomic workflow.

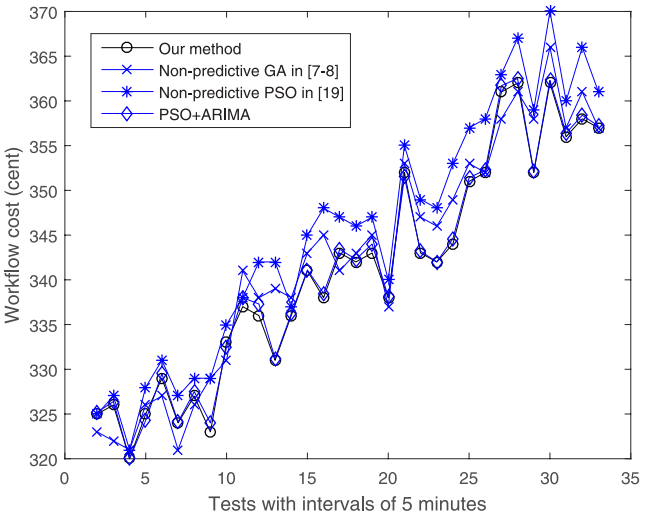


FIGURE 26. Comparison of cost of Epigenomics workflow.

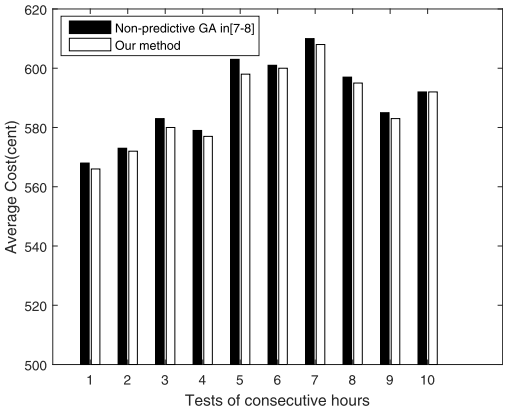


FIGURE 27. Comparison of cost of Montage in consecutive hours.

We test these VMs by using a Sugon I450 server (4-CPU Intel Xeon 5506/128G RAM)) in the period between 8:00AM to 10:40AM on May 17th, 2017 and obtain a series of execution time of different types of VMs to calculate (using the GaussLegendre algorithm) circumference ratio reaching a

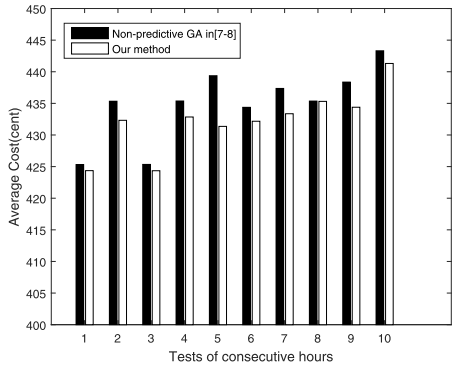


FIGURE 28. Comparison of cost of Cybershake in consecutive hours.

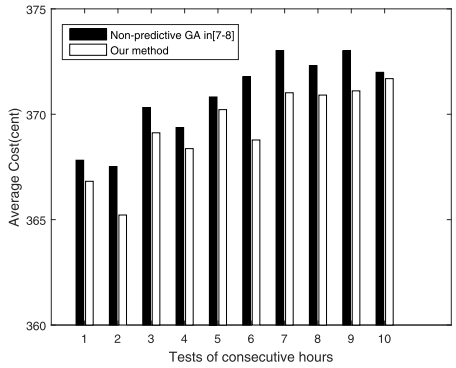


FIGURE 29. Comparison of cost of Epigenomics in consecutive hours.

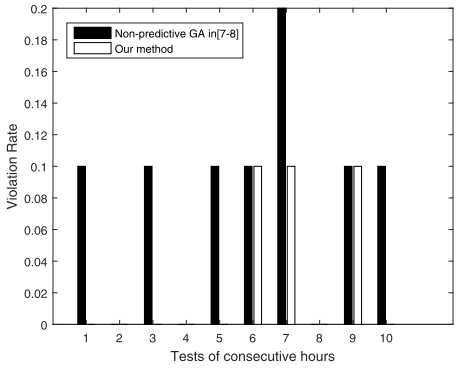


FIGURE 30. Comparison of violation rate of Montage in consecutive hours.

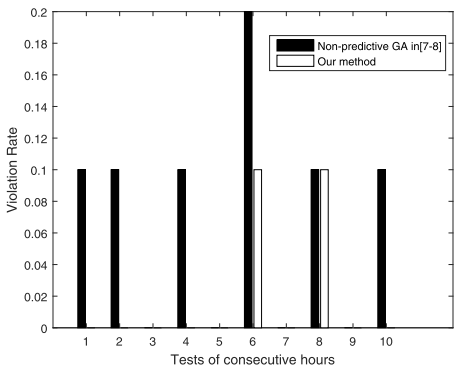


FIGURE 31. Comparison of violation rate of Cybershake in consecutive hours.

varying number of decimal digits, e.g., 8, 16 and 32 million, with a constant interval of 5 minutes, as shown in Figs. 18-20 and earlier in Figs. 1-9. These results suggest that run-time

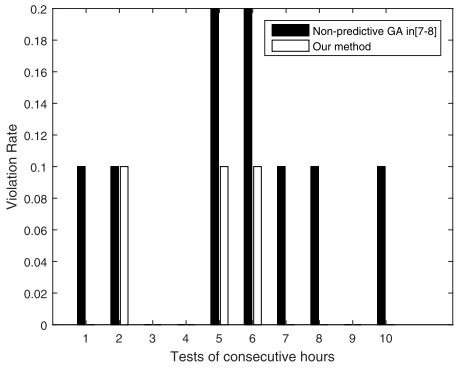


FIGURE 32. Comparison of violation rate of Epigenomics in consecutive hours.

performance of commercial clouds is indeed time-varying and unstable. Note that we consider a constant interval of 5 minutes simply because tests show that it usually takes no more than 300 seconds to run sample workflows. 5 minutes is thus considered to be a safe interval to avoid the case that a new trial of the scheduling algorithm is initiated before preceding ones are accomplished.

We also use another Sugon I450 server within the same local-area-network to yield schedules by executing GA every 5 minutes (since the time interval for ARIMA performance series is 5 minutes and a schedule is thus generated within the identical time interval). Such generated schedules are fluctuation-aware and aim at guaranteed performance and reduced cost. For a comparison purpose, we also apply a representative non-predictive algorithm proposed in [7] and [8] to schedule three workflows. Note that we consider these two as the baseline algorithms because: 1) their physical model formulation is identical to ours; 2) their test results suggest that they outperform all other earlier methods; and 3) their proposed algorithms are different in details but actually equally effective. As shown in Fig. 21-26, our proposed method achieves less cost (by 1.1231 cents for Montage, 0.6670 cents for Cybershake, and 1.4002 cents for Epigenomics) in average and lower SLA violation rates as well (6.06% vs. 12.90% for Montage, 0 vs. 3.23% for Cybershake, and 3.23% vs. 9.68% for Epigenomics)). It can be observed that our proposed method outperforms a PSO-based approach proposed in [18] and [19] and even the combination of PSO with ARIMA (although no existing contribution does so). It is interesting to see that the non-predictive GA method in [7] and [8] clearly outperforms the PSO-based one. Intuitively, the disadvantage of a non-predictive approach lies in that it ignores up/down performance trends of VMs. It therefore tends to choose expensive VMs when inexpensive VMs have satisfactory and improved future performance, thereby increasing its cost. It also tends to choose inexpensive VMs with satisfactory current performance when such VMs have worsened future performance and thus leads to longer workflow completion time and higher SLA violation rates. To further show the effectiveness of our proposed method, we illustrate comparisons of cost and SLA violation rates in consecutive

hours after 10:40AM on May 17th, 2017. It can be seen that the advantage of our proposed method is evident.

VIII. CONCLUSIONS AND FURTHER STUDIES

In this work, we introduce a comprehensive framework for optimal scientific workflow scheduling on IaaS clouds. Instead of assuming constant or bounded performance of VMs as most existing methods do, our proposed method is capable of modeling time-varying performance and working out cost-effective schedules to reduce workflow cost while fulfilling Service-Level-Agreement (SLA). A case study based on real-world third-party IaaS clouds and some well-known scientific workflows show that our proposed approach outperforms traditional approaches that consider time-invariant or bounded VM performance only.

We plan to consider the following topics for future work: 1) More quantitative metrics, e.g., fault tolerance and cloud mobility, are supposed to be analyzed and optimized; 2) Petri nets [31]–[33] can be borrowed as a modeling formalism, where structural reduction techniques can be employed to model fine-grained control-flow activities of scientific workflows deployed on clouds; 3) this work consider hard SLA constraints. We intend to consider soft ones (where workflow completion time is allowed to exceed a threshold value with a bounded given rate) and introduce corresponding algorithms to generate run-time schedules; 4) our proposed method relies on knowledge of time-series data of all tasks and candidate cloud servers. However, in some cases it is not feasible to collect such data at run-time. We thus intend to introduce large-scale-sparse-matrices-analysis models [34] [35] for performance prediction when historical data is missing; 5) We intend to introduce updated designs of encoding, mutation, crossover of the genetic algorithm to further improve its performance [36]; and 6) We intend to consider mobile opportunistic networks as the supporting platforms for scientific workflows and introduce corresponding predictive scheduling algorithms.

REFERENCES

[1] W. Tan and M. C. Zhou, *Business and Scientific Workflows: A Web Service-Oriented Approach*. Hoboken, NJ, USA: Wiley, 2013.

[2] 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. Automatica Sinica*, vol. 4, no. 1, pp. 6–18, Jan. 2017.

[3] M. A. Rodriguez and R. Buyya, "A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments," *Concurrency Comput., Pract. Exper.*, vol. 29, no. 8, p. e4041, 2017.

[4] J. Schad, J. Dittrich, and J. A. Quiané-Ruiz, "Runtime measurements in the cloud: Observing, analyzing, and reducing variance," *Proc. VLDB Endowment*, vol. 3, nos. 1–2, pp. 460–471, Sep. 2010.

[5] K. R. Jackson *et al.*, "Performance analysis of high performance computing applications on the Amazon Web services cloud," in *Proc. 2nd IEEE Int. Conf. Cloud Comput. Technol. Sci.*, Nov./Dec. 2010, pp. 159–168.

[6] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *J. ACM*, vol. 24, no. 2, pp. 280–289, Apr. 1977.

[7] J. Meena, M. Kumar, and M. Vardhan, "Cost effective genetic algorithm for workflow scheduling in cloud under deadline constraint," *IEEE Access*, vol. 4, pp. 5065–5082, 2016.

[8] Z. Zhu, G. Zhang, M. Li, and X. Liu, "Evolutionary multi-objective workflow scheduling in cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 5, pp. 1344–1357, May 2016.

[9] L. Liu, M. Zhang, R. Buyya, and Q. Fan, "Deadline-constrained coevolutionary genetic algorithm for scientific workflow scheduling in cloud computing," *Concurrency Comput., Pract. Exper.*, vol. 29, no. 5, p. e3942, Mar. 2017.

[10] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proc. Int. Conf. High Perform. Comput. Netw., Storage Anal.*, Nov. 2011, Art. no. 49.

[11] N. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds," in *Proc. Int. Conf. High Perform. Comput. Netw., Storage Anal.*, May 2012, Art. no. 22.

[12] S. Abrishami, M. Naghibzadeh, and D. H. J. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Gener. Comput. Syst.*, vol. 29, no. 1, pp. 158–169, Jan. 2013.

[13] R. N. Calheiros and R. Buyya, "Meeting deadlines of scientific workflows in public clouds with tasks replication," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 7, pp. 1787–1796, Jul. 2014.

[14] D. Poola, S. K. Garg, R. Buyya, Y. Yang, and K. Ramamohanarao, "Robust scheduling of scientific workflows with deadline and budget constraints in clouds," in *Proc. 28th Int. Conf. Adv. Inf. Netw. Appl. (AINA)*, May 2014, pp. 858–865.

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

[16] E.-K. Byun, Y.-S. Kee, J.-S. Kim, and S. Maeng, "Cost optimized provisioning of elastic resources for application workflows," *Future Gener. Comput. Syst.*, vol. 27, no. 8, pp. 1011–1026, Oct. 2011.

[17] H. Wu, X. Hua, Z. Li, and S. Ren, "Resource and instance hour minimization for deadline constrained DAG applications using computer clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 3, pp. 885–899, Mar. 2016.

[18] S. Pandey, L. Wu, S. M. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in *Proc. 24th IEEE Int. Conf. Adv. Inf. Netw. Appl.*, Apr. 2010, pp. 400–407.

[19] M. A. Rodriguez and R. Buyya, "Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds," *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 222–235, Apr./Jun. 2014.

[20] Z.-G. Chen, K.-J. Du, Z.-H. Zhan, and J. Zhang, "Deadline constrained cloud computing resources scheduling for cost optimization based on dynamic objective genetic algorithm," in *Proc. IEEE Congr. Evol. Comput.*, May 2015, pp. 708–714.

[21] X. Yin, X. Ma, and K. S. Trivedi, "An interacting stochastic models approach for the performance evaluation of DSRC vehicular safety communication," *IEEE Trans. Comput.*, vol. 62, no. 5, pp. 873–885, May 2013.

[22] R. Ghosh, F. Longo, F. Frattini, S. Russo, and K. S. Trivedi, "Scalable analytics for IaaS cloud availability," *IEEE Trans. Cloud Comput.*, vol. 2, no. 1, pp. 57–70, Jan. 2014.

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

[24] F. Dong, J. Luo, and B. Liu, "A performance fluctuation-aware stochastic scheduling mechanism for workflow applications in cloud environment," *IEICE Trans. Inf. Syst.*, vol. E97.D, no. 10, pp. 2641–2651, May 2014.

[25] H. Ma, H. Zhu, Z. Hu, W. Tang, and P. Dong, "Multi-valued collaborative QoS prediction for cloud service via time series analysis," *Future Gener. Comput. Syst.*, vol. 68, pp. 275–288, Mar. 2017.

[26] J. Li, X. Luo, Y. Xia, Y. Han, and Q. Zhu, "A time series and reduction-based model for modeling and QoS prediction of service compositions," *Concurrency Comput., Pract. Exper.*, vol. 27, no. 1, pp. 146–163, Jan. 2015.

[27] Y. Xia, M. Zhou, X. Luo, S. Pang, and Q. Zhu, "Stochastic modeling and performance analysis of migration-enabled and error-prone clouds," *IEEE Trans. Ind. Informat.*, vol. 11, no. 2, pp. 495–504, Apr. 2015.

[28] Y. Xia, M. Zhou, X. Luo, Q. Zhu, J. Li, and Y. Huang, "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.

[29] Q. Wu, Q. Zhu, X. Jian, and F. Ishikawa, "Broker-based SLA-aware composite service provisioning," *J. Syst. Softw.*, vol. 96, pp. 194–201, Oct. 2014.

[30] D. Whitley, "A genetic algorithm tutorial," *Statist. Comput.*, vol. 4, no. 2, pp. 65–85, Jun. 1994.

[31] Z. Ding, C. Jiang, and M. Zhou, "Design, analysis and verification of real-time systems based on time Petri net refinement," *ACM Trans. Embedded Comput. Syst.*, vol. 12, no. 1, pp. 4:1–4:18, Jan. 2013.

[32] N. Ran, H. Su, and S. Wang, "An improved approach to test diagnosability of bounded Petri nets," *IEEE/CAA J. Autom. Sinica*, vol. 4, no. 2, pp. 297–303, Apr. 2017.
[33] Y. Chen, Z. Li, and M. Zhou, "Optimal supervisory control of flexible manufacturing systems by Petri nets: A set classification approach," *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 2, pp. 549–563, Apr. 2014.
[34] X. Luo, M. Zhou, M. Shang, S. Li, and Y. Xia, "A novel approach to extracting non-negative latent factors from non-negative big sparse matrices," *IEEE Access*, vol. 4, pp. 2649–2655, 2016.
[35] X. Luo, M. C. Zhou, S. Li, Z. You, Y. Xia, and Q. Zhu, "A nonnegative latent factor model for large-scale sparse matrices in recommender systems via alternating direction method," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 3, pp. 579–592, Mar. 2016.
[36] T. Mareda, L. Gaudard, and F. Romerio, "A parametric genetic algorithm approach to assess complementary options of large scale wind-solar coupling," *IEEE/CAA J. Autom. Sinica*, vol. 4, no. 2, pp. 260–272, 2017.
[37] Y. Hou, N. Wu, M. C. Zhou, and Z. Li, "Pareto-optimization for scheduling of crude oil operations in refinery via genetic algorithm," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 47, no. 3, pp. 517–530, Mar. 2017.

WEILING LI received the B.S. and M.S. degrees in software engineering from Chongqing University in 2010 and 2014, respectively, where he is currently pursuing the Ph.D. degree in software engineering with the Key Laboratory of Software Theory and Technology. His research interests are in Cloud computing, workflows, and software quality.

YUNNI XIA (SM'14) received the B.S. degree in computer science from Chongqing University, China, in 2003, and the Ph.D. degrees in computer science from Peking University, China, in 2008. Since 2008, he has been a Professor with the School of Computer Science, Chongqing University. He has authored or co-authored over 50 research publications. His research interests are in Petri nets, software quality, performance evaluation, and cloud computing system dependability.

MENGCHU ZHOU (S'88–M'90–SM'93–F'03) received the B.S. degree from the Nanjing University of Science and Technology, Nanjing, China in 1983, the M.S. degree from the Beijing Institute of Technology, Beijing, China, in 1986, and the Ph.D. degree from Rensselaer Polytechnic Institute, Troy, NY, USA, in 1990. He joined New Jersey Institute of Technology (NJIT), Newark, NJ, USA, in 1990, where he is a Distinguished Professor of Electrical and Computer Engineering and the Director of the Discrete-Event Systems Laboratory. He has over 700 publications, including 12 books, 400 journal papers (over 300 in IEEE transactions), and 28 book-chapters. He has 11 patents and several pending ones. His research interests are in intelligent automation, Petri nets, Internet of Things, Web service, workflow, big data, transportation and energy systems.

He was invited to lecture in Australia, Canada, China, France, Germany, Hong Kong, Italy, Japan, Korea, Mexico, Qatar, Saudi Arabia, Singapore, Taiwan, and USA, and served as a Plenary/Keynote Speaker for many conferences. He is a Life Member of the Chinese Association for Science and Technology-USA and served as its President in 1999. He is fellow of the International Federation of Automatic Control, American Association for the Advancement of Science, and Chinese Association of Automation. He is also the VP for Conferences and Meetings of IEEE SMC Society. He has been among most highly cited scholars for years and ranked top one in the field of engineering worldwide in 2012 by Web of Science/Thomson Reuters and now Clarivate Analytics. He was a recipient of the NSF's Research Initiation Award, the CIM University-LEAD Award from Society of Manufacturing Engineers, the Perlis Research Award and Fenster Innovation in Engineering Education Award from NJIT, the Humboldt Research Award for US Senior Scientists from Alexander von Humboldt Foundation, the Leadership Award and Academic Achievement Award from Chinese Association for Science and Technology-USA, the Asian American Achievement Award from Asian American Heritage Council of New Jersey, and Outstanding Contributions Award, 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 is the founding Co-Chair of the Enterprise Information Systems Technical Committee (TC) and Environmental Sensing, Networking, and Decision-making TC of IEEE SMC Society. He was the General Chair of the IEEE Conference on Automation Science and Engineering, Washington D.C., USA, in 2008, the General Co-Chair of the 2003 IEEE International Conference on System, Man and Cybernetics (SMC), Washington DC, USA, in 2003, the Founding General Co-Chair of the 2004 IEEE international conference on Networking, Sensing and Control, Taipei, in 2004, and the General Chair of the 2006 IEEE international conference on Networking, Sensing and Control, Ft. Lauderdale, FL, USA, in 2006. He was the Program Chair of the 2010 IEEE International Conference on Mechatronics and Automation, in 2010, Xi'an, China, the 1998 and 2001 IEEE International Conference on SMC and 1997 IEEE International Conference on Emerging Technologies and Factory Automation. He organized and chaired over 100 technical sessions and served on program committees for many conferences. He has led or participated in over 50 research and education projects with total budget over 12*M*, funded by National Science Foundation, Department of Defense, NIST, New Jersey Science and Technology Commission, and industry. He is the founding Editor of the IEEE Press Book Series on Systems Science and Engineering and the Editor-in-Chief of the IEEE/CAA Journal of Automatica Sinica. He served as an Associate Editor of IEEE Transactions on Robotics and Automation, the IEEE Transactions on Automation Science and Engineering, the IEEE Transactions on Systems, Man and Cybernetics: Systems, and the IEEE Transactions on Industrial Informatics, and the Editor of the IEEE Transactions on Automation Science and Engineering. He is also an Associate Editor of the IEEE Transactions on Intelligent Transportation Systems, the IEEE Internet of Things Journal, and *Frontiers of Information Technology & Electronic Engineering*.

XIAONING SUN received the B.S. degree in computer science from Chongqing University, China, in 2015, where she is currently pursuing the Ph.D. degree with the College of Computer Science. Her current research interests include performance evaluation, heterogeneous cloud services, and scientific workflows.

QINGSHENG ZHU received the B.S., M.S., and Ph.D. degrees in computer science from Chongqing University, Chongqing, China, in 1982, 1985, and 2000, respectively. He was a Visiting Scholar with the University of London, London, U.K., from 1993 to 1994, and also a Visiting Professor with the University of Illinois, Chicago, IL, USA, from 2001 to 2002. He is currently a Professor with the Computer College, and the Director of the Key Laboratory of Software Theory and Technology, Chongqing University. His research interests include services computing and data mining.