Zhu, J.; Li, X.; Ruiz García, R.; Xu, X. (2018). Scheduling Stochastic Multi-Stage Jobs to Elastic Hybrid Cloud Resources. IEEE Transactions on Parallel and Distributed Systems. 29(6):1401-1415. https://doi.org/10.1109/TPDS.2018.2793254

Scheduling Stochastic Multi-stage Jobs to Elastic Hybrid Cloud Resources

Jie Zhu, Xiaoping Li, *Senior Member, IEEE,* Rubén Ruiz, Xiaolong Xu

Abstract—We consider a special workflow scheduling problem in a hybrid-cloud-based workflow management system in which tasks are linearly dependent, compute-intensive, stochastic, deadline-constrained and executed on elastic and distributed cloud resources. This kind of problems closely resemble many real-time and workflow-based applications. Three optimization objectives are explored: number, usage time and utilization of rented VMs. An iterated heuristic framework is presented to schedule jobs event by event which mainly consists of job collecting and event scheduling. Two job collecting strategies are proposed and two timetabling methods are developed. The proposed methods are calibrated through detailed designs of experiments and sound statistical techniques. With the calibrated components and parameters, the proposed algorithm is compared to existing methods for related problems. Experimental results show that the proposal is robust and effective for the problems under study.

Index Terms—Multi-stage job scheduling, Linearly dependent tasks, Stochastic, Deadline-constraint, Elastic, Cloud computing.

1 Introduction

In this paper, we consider a multi-stage job scheduling problem in a hybrid-cloud-based workflow management system (HCWMS). Computational tasks are massive, linearly dependent, stochastic, compute-intensive, deadline-constrained and are executed on elastic and distributed cloud resources. The considered problem involves the execution of massive tasks on multiple cloud platforms, where each task takes a short time (a few seconds) to execute. Tasks are linearly dependent and each set of dependent tasks is taken as a multi-stage job with the same linear processing route. These multi-stage jobs arrive stochastically, which means the arrival and processing times are not known beforehand. Jobs are compute-intensive and data transferring times inside the cloud are negligible. The data transferring latency from one cloud to another cloud however, is not negligible and is studied in this paper. Each job is assigned a hard deadline. Multiple computing resources (e.g., virtual machines, VMs) are provisioned for processing each stage of the involved jobs and computing resources can scale elastically across multiple cloud platforms at run time.

- *Jie Zhu works at the department of Computer Science & Technology, Nanjing University of Posts & Telecommunications, China, 210048; Institute of Big Data Research, Nanjing University of Posts & Telecommunications, Yancheng, China; and Jiangsu Key Laboratory of Big Data Security & Intelligent Processing, Nanjing University of Posts and Telecommunications, China, 210023.*
  E-mail: zhujie@njupt.edu.cn
- *Xiaoping Li works at the School of Computer Science and Engineering, Southeast University, Nanjing, China, 211189; and also at the Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education, Nanjing, China, 211189.*
  E-mail: xpli@seu.edu.cn
- *Rubén Ruiz is with Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Edifico 8G, Acc. B. Universitat Politècnica de València, Camino de Vera s/n, 46021, València.*
  E-mail: rruiz@eio.upv.es
- *Xiaolong Xu works at the department of Computer Science & Technology, Nanjing University of Posts & Telecommunications,China,210048.*
  E-mail: xuxl@njupt.edu.cn

Manuscript revised December 16, 2017.

These workflows are widespread in many real-time workflow applications [1], [2]. For example, in real-time image processing, object recognition contains five stages. Each stage is a task. All tasks (Gray scale, sobel, Gaussian blur, fast corner and SAD matching) are processed sequentially [3]. These tasks are compute-intensive. Each end user's request for the application is regarded as a 5-stage job in the HCWMS. Users' requests arrive stochastically. To improve user experience, the HCWMS assigns a deadline to each job in terms of [4]. To avoid bottlenecks, the HCWMS deploys the application on 5 virtual clusters to deal with these 5 tasks separately. Each cluster initially contains multiple on-premise VMs. Clusters scale up at run time by renting VMs from public clouds only when on-premise VMs cannot handle the workload.

There are many studies on scheduling problems in cloud computing. Such problems are NP-hard most of the time [5]. However, the considered problem is different from the traditional ones in several aspects: (i) Usually resources are assumed to be fixed in quantity [6] and in this paper we consider that they are scalable in hybrid cloud platforms. (ii) Traditional cloud scheduling problems usually involve soft deadlines within limited resources of physical clusters [7]–[9] in which a penalty is charged for tardiness. The trade-off between penalty and renting costs is to be balanced. In practice, users prefer on-time applications rather than a compensation, i.e., hard deadlines result in a better user experience. Scalable resources in hybrid clouds make it possible to meet hard deadlines by renting necessary resources from public clouds. Therefore, we consider hard deadlines in this paper. (iii) Time delay [10], costs [11], makespan [12], time-cost trade-off [13] and energy consumption [14] are commonly optimized. The objective to optimize in this paper is relative to the renting cost of VMs. There are many VM pricing structures on public cloud platforms, for example, on-demand instance pricing and reserved instance pricing. With the same pricing structure the prices of VMs are

different on different cloud platforms. In order not to bind the considered problem to a specific pricing, we do not optimize renting cost directly. Three metrics are employed to evaluate the proposed methods: the total usage time, the number and the utilization of rented VMs.

Since batches of jobs arrive at HCWMS stochastically, dynamic amounts of resources are required in order to carry out the tasks (which can be up to millions). Existing methods, such as round robin, Random-Neural-Network based reinforcement learning [15], sensible decision [16], adaptive dispatching [17], max-min cloud algorithm [18], linear-programming-based affinity scheduling [19], the PSO-based algorithm [20] and the genetic algorithm [21], are proposed for either static or special dynamic task scheduling problems. It is desirable to develop effective and efficient scheduling methods for massive tasks with precedences and dynamic resource requirements.

Stochastically arriving jobs imply that the required workload may exceed the capacity of the currently available VMs. In order to meet the hard deadlines, the HCWMS scales applications up to public clouds automatically, i.e., rented VMs are added to alleviate the overloaded virtual clusters. It is assumed that on-premise VMs are free of charge in the HCWMS and the rented ones are not. Therefore, it is necessary to propose scheduling methods for the considered problem in the HCWMS trying to fully utilize on-premise VMs and pay for rented VMs as little as possible. However, it is difficult to develop scheduling methods for massive tasks with hard deadlines and minimal operating costs.

In practical environments, the workloads of applications are usually unpredictable and differ in time requirements. Adaptive algorithms [22] offer robust schemes with the potential to perform better for some workload scenarios. However, no single algorithm is effective for all possible workload scenarios. A more reasonable and resilient way is to execute scheduling algorithms adaptively. As the gateway of applications, the HCWMS monitors realtime workload requirements and selects the most suitable scheduling method from a set of candidates in response to the dynamic workload requirement. For such an adaptive HCWMS, it is necessary to evaluate the performance of different scheduling methods in possible workload scenarios and to calibrate the involved parameters over massive experimental instances.

The primary aim of the paper is to provide an efficient, effective and adaptive method for the stochastic multi-stage job scheduling problem in HCWMS. The main contributions of this paper are summarized as follows.

(i) A new multi-stage job scheduling problem is investigated for workflows in hybrid clouds in which massive, linearly dependent, stochastic, compute-intensive and hard deadline-constrained tasks are scheduled to elastic and distributed cloud resources. The considered problem is more realistic than previously studied settings.

(ii) A dynamic event scheduling algorithm is developed which periodically schedules newly arriving tasks by a greedy method. A re-schedule strategy is introduced to optimize the global objective.

(iii) Idle time slots of VMs are maintained using balanced search trees in which time slots can be quickly allocated to ready tasks. The priorities of ready tasks are dynamically sorted by a min heap to speed up calculations.

The rest of the paper is organized as follows. Related works are reviewed in Section 2. Section 3 describes and formalizes the problem under study. A heuristic is proposed for the considered problem in Section 4. Section 5 evaluates the performance of the proposal under different workload scenarios followed by conclusions in Section 6.

2 Related Work

With the development of hybrid cloud and PaaS (Platform as a Service), cloud workflow management systems deploy applications not only to on-premise VMs but also to VMs rented from multiple public cloud platforms. Through web service APIs (provided by public cloud platforms or cloud brokers), the HCWMS provisions scientific/business applications automatically on rented VMs and obtains peak load capacity by outsourcing the workload on these VMs [23]. Many frameworks (broker-based, agent-based, etc.) have been proposed to support the execution of workflow management in hybrid clouds [24]–[27]. The considered special workflow scheduling problem is the core of the HCWMS framework. Workflow scheduling problems with different constraints and objectives have been widely studied in the literature. Both static and dynamic workflow scheduling problems have been solved by heuristics and meta-heuristics.

Many meta-heuristic scheduling algorithms have been applied to workflow scheduling problems, such as PSO (Particle Swarm Optimization), GA (Genetic Algorithm), SA (Simulated Annealing) and ACO (Ant Colony Optimization). Tao et al. [28] proposed a rotary chaotic PSO to optimize trustworthy workflow scheduling in a multi-dimensional complex space. Jena [29] used a multi-objective nested PSO to optimize energy and processing time. Liu et al. [30] improved the basic PSO with a variable neighborhood for security-constraint and data-intensive workflow scheduling. Cui et al. [31] proposed the GA based data replica placement strategy for data related workflow scheduling with the objective of minimizing data transmissions. Ahmad et al. [32] enhanced the GA by modifying genetic operators and by including an efficient heuristic which adapted a heuristic to generate the initial population. The proposed algorithm can optimize the load balancing during the execution to utilize resources at maximum capacity. Lopez-Garcia et al. [33] combined GA with a Cross Entropy (CE) method. The algorithm divided the population into two sub-populations in order to apply GA in one sub-population and CE in the other. Eawna et al. [34] proposed a hybrid algorithm combining PSO and SA, and showed that resource provisioning based on the PSO-SA algorithm is much faster than the PSO algorithm and the SA algorithm. Kianpisheh et al. [35] employed an ant colony system to minimize an aggregation of reliability and constraints violation. Three novel heuristics were proposed which are adaptively selected by ants. Two of them were employed to find feasible schedules and the other was

used to enhance reliability. Most of these meta-heuristics perform well on static workflow scheduling. However, they need considerable amounts of CPU time to provide high quality solutions, and this is hardly acceptable in real cloud computing settings.

Fast heuristics are able to solve dynamic scheduling problems with a low overhead. Many scheduling systems employ fast heuristics to run millions of tasks. Ousterhout et al. [36] adapted a good load balancing strategy with near optimal performance using a randomized sampling approach. Schwarzkopf et al. [37] presented three different simple policies for allocating resources: max-parallelism, global cap and relative job size, among which the resource allocation solution leading to the earliest possible finish time was selected. Rajendran et al. [38] used an adaptive job stealing approach that makes the system highly scalable and dynamic. Sadooghi et al. [39] developed a cloud-enabled distributed task execution framework, in which distributed queues were used to deliver the tasks fairly to processors. Wang et al. [40]–[42] implemented a smart workload allocation platform TAP which applies both model based and learning based approaches, exploits measurements collected in real time in the cloud, and dynamically make task allocation decisions that optimize QoS. Bertin et al. [43] applied Lagrangian optimization and distributed gradient descent for fairly scheduling concurrent Bag-of-Tasks (BoT) applications. Benoit et al. [44] designed an optimal algorithm for the offline BoT scheduling and adapted it to an online scenario. In addition, some works modified classical heuristics in order to adapt to scheduling problems in clouds, such as round robin, Random-Neural-Network based reinforcement learning [15], sensible decision [16], adaptive dispatching [17], max-min cloud algorithm [18] and linear-programming-based affinity scheduling [19], etc. All these systems and algorithms aim to process millions of tasks per second. They employ simple and fast approaches for independent task scheduling and resource allocating. However, they are not efficient for scheduling problems with dependency. Due to the precedence constraints caused by the tasks's dependency, the time for computing a feasible schedule increases significantly. Heterogeneous Earliest Finish Time (HEFT) [45] is the most popular list based heuristic for dynamic workflow scheduling. It orders dependent tasks based on priorities and then assigns them to suitable resources to achieve high performance. In addition, Min-min, Max-min and Critical-Path-on-a-processor (CPOP) [45] are common heuristics for the dynamic workflow scheduling.

Recently, elastic/scalable scheduling in cloud environments has attracted more and more attention. To determine the number of elastic resources some existing research employs the batch-queue model [46], [47]. Delicate triggers and thresholds have been designed and long/short-term behaviors of elastic instances studied. Some investigations focused on elastic workflows in hybrid clouds. Bittencourt and Madeira [48] proposed the Time and Cost Optimization for Hybrid Clouds (TCHC) algorithm to reduce the execution time and costs of multiple workflows. Liu et al. [49] proposed a multi-objective scheduling method for a workflow on a multi-site cloud. The proposed algorithm consisted of a Single Site Virtual Machine Provisioning approach (SSVP) and ActGreedy, a multisite scheduling approach. Fard et al. [50] introduced a pricing model and a truthful mechanism considering monetary cost and completion time objectives. Gelenbe et al. [51] studied the optimum load sharing between a local and remote cluster service with the objective of minimizing the average response time and energy consumption per job, where the task dependency is not considered. Duan et al. [52] formulated the BoT scheduling problem as a new sequential cooperative game and proposed a communication and storage-aware multi objective algorithm to optimize the execution time and the economic cost. However, the proposed method is designed for the off-line BoT scheduling problem which can be taken as the specific static case of the considered problem. In our previous work [53], [54], we investigated the static case of the considered problem and proposed a local search heuristic. However, dynamic scheduling problems with the network delay are much closer to reality. In another previous work [55], we investigated the considered problem with the objective of minimizing the number of rented VMs. The data transfer delays are not considered.

The above mentioned problems can be classified into three types: (i) Scheduling problems in manufacturing in which resources are not elastic and distributed. (ii) Dynamic non-dependent task scheduling problems in cloud environments. (iii) Static scheduling problems considering both elasticity and task dependency. To the best of our knowledge, the scheduling problem jointly regarding all the above mentioned features and the objectives has not been studied yet.

3 Problem Description

3.1 Problem Scenario

There are three aspects in the considered scenario: cloud service provider (CSP), HCWMS and end users. The CSP uploads the deployment package of its application to the HCWMS. The deployment package includes installation files for the application and guidelines which support the management of the application by the HCWMS. Guidelines are constructed based on the standards that the HCWMS parses and understands, e.g., the TOSCA [56]. In the guidelines, the CSP describes the processing route of the jobs and the hard deadline constraint, i.e., deadline misses are strictly forbidden. Guidelines also include some instructions which include how to deploy the application, how to scale it and how to estimate the processing time of each task of a job, etc. The HCWMS receives the deployment package and deploys the application according to the guidelines. For cost considerations, it initially deploys the application on on-premise VMs from an owned data center. When the application is deployed and available for end users, they submit requests to the HCWMS for the application with information about their demands, such as expected response times, QoS parameters and if expedited services are needed or not. Once the HCWMS receives requests from end users (as depicted in Fig. 1): (i) it maps them to a set of multi-stage linear jobs; (ii) it estimates the processing times of these jobs and assigns deadlines to them based on end users' demands; (iii) it schedules jobs on available VMs feasibly, effectively and efficiently, which is exactly the aim of this paper.

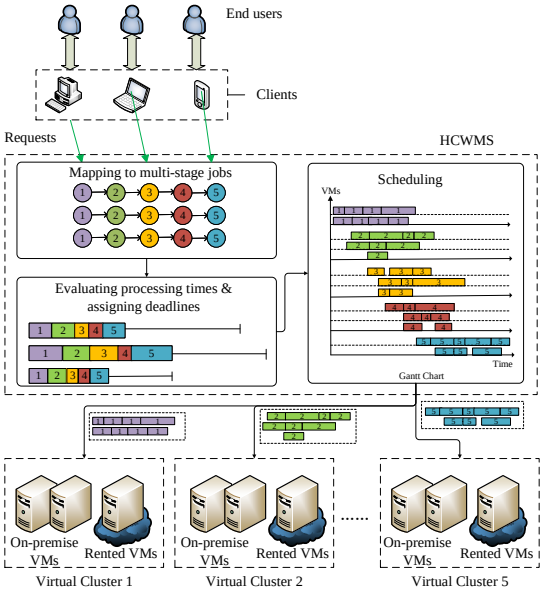


Fig. 1. The procedure in HCWMS for processing requests

3.2 Application & Resource Model

The application model and resource model are described as follows (important notations used for the description are given in Table 1). A massive set of jobs arrives at the system stochastically. In terms of [57], the set of jobs arriving during a small time period t is called a JRE (Job-related Real-time Event). At every time period t, the newly arriving jobs are scheduled. Suppose there are Q JREs (denoted by $\mathbb{E} = (E_1, \ldots, E_Q)$) arriving during the time period $[0, t \times Q]$. At the q^{th} JRE $E_q =< \alpha_q, \mathbb{J}_q >$, a set of jobs $\mathbb{J}_q$ arrives at time $\alpha_q = t \times (q-1)$ (actually, they arrive in $[t \times (q-1),\ t \times q)$). Suppose n_q is the number of jobs in the q^{th} JRE, i.e., $n_q = |\mathbb{J}_q|$. Therefore, $n = \sum_{q=1}^{Q} n_q$ jobs arrive at the system during $[0, t \times Q]$. Job J_j $(j = 1, \ldots, n)$ is composed of m tasks (or m stages) $\mathcal{T}_{j,1}, \mathcal{T}_{j,2}, \ldots, \mathcal{T}_{j,m}$ which are provisioned by m virtual clusters. Each task $\mathcal{T}_{j,i}$ $(i = 1, \ldots, m)$ is processed on one virtual cluster which includes a set of VMs as a VM container. There are two types of VMs in virtual clusters: on-premise VMs and rented VMs. On-premise VMs are those deployed in the local data center while rented VMs come from public clouds. Let V_i be the virtual cluster processing the i^{th} task $\mathcal{T}_{j,i}$ $(i = 1, 2, \ldots, m)$ of each job J_j. v_i^k denotes the k^{th} VM of V_i. The number of on-premise VMs o_i in V_i is constant. The number of rented VMs r_i increases with workload. All VMs in V_i are indexed by $V_i = (v_i^1, v_i^2, \ldots, v_i^{o_i+r_i})$ where $v_i^1, \ldots, v_i^{o_i}$ are on-premise VMs and the remaining VMs $(v_i^{o_i+1}, \ldots, v_i^{o_i+r_i})$ are rented.

Each job J_j $(j = 1, 2, \ldots, n)$ follows the same exact processing route across all tasks starting from $\mathcal{T}_{j,1}$ to $\mathcal{T}_{j,m}$. A job can be taken as a special workflow as shown in Fig. 2 (a) with the linear processing route. Jobs in the same event can also be taken as a workflow as shown in Fig. 2 (b) if a dummy start task and a dummy end task are added.

In this paper, each VM is an implementation of an application execution environment such as a Java virtual machine (JVM). Multi-tenant scenarios are not considered, i.e., each VM can only process one task at a time. In order to facilitate the management of VMs, VMs in the same cluster are provisioned with identical configurations, i.e., VMs are homogeneous in the same cluster.

TABLE 1
Notations used for the problem description

Notation	Description
t	Time period for scheduling
$\mathbb{E}$	Set of Events
Q	Number of real-time events in $\mathbb{E}$
n	Number of jobs arriving during $[0, t \times Q]$
m	Number of stages in a job;
j, j'	Identifier of jobs
i	Identifier of stages;
k	Identifier of VMs
q	Identifier of events
J_j	j^{th} job, $j = 1, ..., n$
E_q	q^{th} event in $\mathbb{E}$
α_q	Time that E_q takes place
$\mathbb{J}_q$	Set of jobs in E_q
n_q	Number of jobs in E_q
$\mathcal{T}_{j,i}$	i^{th} task of J_j
V_i	i^{th} virtial cluster
v_i^k	k^{th} VM of V_i
o_i	Number of on-premise VMs in V_i
r_i	Number of rented VMs in V_i
$p_{j,i}$	Processing time of task $\mathcal{T}_{j,i}$
t_j	Arrival time of J_j
$b_{j,i}$	Start time of $\mathcal{T}_{j,i}$
$c_{j,i}$	Completion time of $\mathcal{T}_{j,i}$
$d_{j,i}$	Transferring delay for the intermediate result of $\mathcal{T}_{j,i}$
D_j	Deadline of J_j
$C_{j,i}$	Latest completion time of $\mathcal{T}_{j,i}$
$v_{j,i}$	VM allocated to $\mathcal{T}_{j,i}$
$< \mathcal{T}_{j,i}, b_{j,i}, v_{j,i} >$	Allocation of $\mathcal{T}_{j,i}$
S	Set of allocations
$F_1(S)$	Number of rented VMs in S
$F_2(S)$	Total usage time of rented VMs in S
$F_3(S)$	Metric for the rented VM utilization in S
$U(v_i^k)$	Utilization ratio of v_i^k
$L(v_i^k)$	Usage time of v_i^k

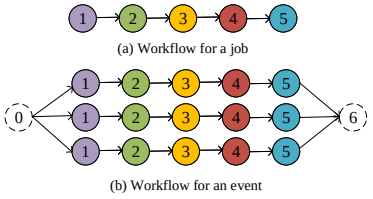


Fig. 2. Workflow for a job

We use t_j $(t_j \geq 0)$ to denote the arrival time of J_j. The processing time of task $\mathcal{T}_{j,i}$ of J_j is $p_{j,i}$ which can be estimated according to guidelines from the CSP or by employing statistical forecasting strategies (e.g.,time-series pattern based interval forecasting strategy in [58]). The processing time is machine-independent since all VMs are homogeneous. $b_{j,i}$ and $c_{j,i}$ denote the start and completion time of $\mathcal{T}_{j,i}$ respectively.

There are three types of data to transfer for a job: the primary data, the intermediate result and the final result. The request of a job is submitted along with the primary data and the job is ready to be scheduled only when the primary data has been fully received by the HCWMS. If some stages of the job are processed in the remote cloud,

then the primary data should be transferred from the local cloud to the remote cloud, which means the transferring delay for the primary data is non-negligible. However, the transferring delay for the primary data can be ignored by synchronizing the primary data to the remote cloud while the HCWMS is receiving the primary data, wherever the job will be assigned to. $d_{j,i}$ is defined as the transferring delay for the intermediate result transferred from $\mathcal{T}_{j,i}$ to $\mathcal{T}_{j,i+1}$. We employ a dummy task 0 with $b_{j,0} = t_j$, $c_{j,0} = t_j$, $p_{j,0} = 0$ and $d_{j,0} = 0$. $\mathcal{T}_{j,i}$ can start only when the result from $\mathcal{T}_{j,i-1}$ is ready, i.e., $b_{j,i} \geq c_{j,i-1} + d_{j,i-1}$. Since jobs are compute-intensive, the intermediate result to forward between successive tasks is not very large, e.g., less than $128K$ (Kilobytes). Therefore, if successive tasks of a job are assigned to VMs in the same network, $d_{j,i}$ is set to zero. If successive tasks of a job are assigned to VMs in different networks, then $d_{j,i} > 0$. $d_{j,m}$ is taken as the transferring delay for the final result, which is output by the last task $\mathcal{T}_{j,m}$. The request can be responded only when the final result is ready in the local data center. Since jobs are compute-intensive, the final result to forward is not very large, e.g., less than $128K$. $d_{j,m} = 0$ when $\mathcal{T}_{j,m}$ is processed in the local data center. If $\mathcal{T}_{j,m}$ is processed in the remote cloud, then $d_{j,m} > 0$. Let D_j be the deadline of J_j. Because of the deadline constraint, the latest completion time of $\mathcal{T}_{j,i}$ is constrained by $C_{j,i} = D_j - \sum_{f=i+1}^{m+1} p_{j,f}$, i.e., $c_{j,i} \leq C_{j,i}$.

The provisioning time (a few seconds in general) for a rented VM is non-negligible. For static problems where arrival and processing times of jobs are given, provisioning times can be determined in advance. The deadline constraint in dynamic problems would be violated because it is difficult to predict when and how many VMs should be rented. An extreme case is that all available VMs are too busy to handle the coming J_j. J_j has to wait until a newly rented VM is ready. If the provisioning time for that rented VM were more than $C_{j,m} - t_j$, the deadline constraint would be violated. The trigger strategy employed in [46] and [47] is effective for the provisioning time of a rented VM, i.e., VMs are rented and provisioned immediately when the number of idle VMs in a virtual cluster is under a predefined threshold. By adopting the trigger strategy, the provisioning time of a rented VM becomes a non-issue.

3.3 Problem Definition

3.3.1 Problem Constraints

Suppose the assignment of the $\mathcal{T}_{j,i}$ to VM v_i^k with start time $b_{j,i}$ is represented by a 3-tuple $< \mathcal{T}_{j,i}, b_{j,i}, v_{j,i} >$, where $v_{j,i} = v_i^k$. Variable $v_{j,i}$ represents the VM allocated to $\mathcal{T}_{j,i}$. A schedule S is a set of allocations, i.e., $S = \{< \mathcal{T}_{j,i}, b_{j,i}, v_{j,i} > | j = 1, \ldots, n, i = 1, \ldots, m\}$. S is feasible if and only if the following constraints are satisfied.

$$\max\{b_{j,i}, b_{j',i}\} \geq \min\{c_{j,i}, c_{j',i}\} \quad (1)$$
$$j \neq j', v_{j,i} = v_{j',i}$$
$$c_{j,i} = b_{j,i} + p_{j,i} \quad (2)$$
$$c_{j,i-1} + d_{j,i-1} \leq b_{j,i} \quad (3)$$
$$t_j \leq b_{j,i} \quad (4)$$
$$c_{j,m} + d_{j,m} \leq D_j \quad (5)$$
$$\forall < \mathcal{T}_{j,i}, b_{j,i}, v_{j,i} > \in S$$
$$j, j' = 1, \ldots, n, i = 1, \ldots, m$$

Constraint (1) guarantees no overlapping time among tasks on the same VM. Constraint (2) indicates that once a task starts it cannot stop until it completes. Constraint (3) defines the precedence constraint, i.e., a task of a job cannot start until its previous task is completed and the intermediate result is received. Constraint (4) indicates a job cannot start before its arrival time. Constraint (5) indicates the deadline constraint.

3.3.2 Optimization Objectives

Suppose the first job arrives at time 0 and the last job arrives at time T. The objective is to generate a feasible schedule with the minimum renting cost within time period $[0, T]$. There are many VM pricing structures on public cloud platforms, e.g., on-demand and reserved instance pricing. In order for not to bind the considered problem to a specific pricing structure, we do not optimize renting costs directly. For the on-demand instance pricing structure, the best metric is the total usage time of rented VMs. For the reserved instance pricing structure, the best metric is the number of rented VMs. Generally, fully utilizing rented VMs leads to less renting costs. Therefore, three metrics are employed to evaluate a feasible schedule S: the number of rented VMs $F_1(S)$, the total usage time of rented VMs $F_2(S)$, and the utilization of rented VMs $F_3(S)$.

Given a feasible schedule S, the objective values $F_1(S)$, $F_2(S)$ and $F_3(S)$ are computed by Eq. (6), Eq. (7) and Eq. (9), respectively.

$$F_1(S) = \sum_{i=1}^{m} r_i \quad (6)$$
$$F_2(S) = \sum_{i=1}^{m} \sum_{k=o_i+1}^{o_i+r_i} L(v_i^k) \quad (7)$$
$$L(v_i^k) = \max\{c_{j,i} | v_{j,i} = v_i^k\} - \min\{b_{j,i} | v_{j,i} = v_i^k\} \quad (8)$$
$$F_3(S) = \sum_{i=1}^{m} \sqrt{\sum_{k=o_i+1}^{o_i+r_i} (1 - U(v_i^k))^2} \quad (9)$$
$$U(v_i^k) = \frac{\sum_{\forall v_{j,i} = v_i^k} p_{j,i}}{\max\{c_{j,i}\} - \min\{b_{j,i}\}} \quad (10)$$
$$\forall < \mathcal{T}_{j,i}, b_{j,i}, v_{j,i} > \in S$$
$$j = 1, \ldots, n, i = 1, \ldots, m$$

Where $L(v_i^k)$ is the usage time of v_i^k and $U(v_i^k)$ is the utilization ratio of v_i^k. $F_3(S)$ is computed based on the Euclidean distance from point $(1, \ldots, 1)$ to point $(U(v_i^{o_i+1}), \ldots, U(v_i^{o_i+r_i}))$, $i = 1, \ldots, m$. Point $(1, \ldots, 1)$ indicates 100% utilization on all rented VMs. Obviously, a higher utilization ratio on on-premise VMs implies less of a need for rented VMs and a smaller objective value. Furthermore, fully utilizing rented VMs leads to a smaller objective value.

4 Proposed Algorithms

The considered dynamic scheduling is completely reactive [59]. No schedule can be generated in advance. Feasible sub-

schedules are successively generated in real time. The sub-schedule of the q^{th} JRE is $s_q = \{< \mathcal{T}_{j,i}, b_{j,i}, v_{j,i} > | J_j \in \mathbb{J}_q, b_{j,i} \geq \alpha_q\}$. By integrating all sub-schedules together, a feasible schedule represented as $S = s_1 \bigcup s_2 \bigcup \ldots \bigcup s_Q$.

All jobs are scheduled event by event. Scheduling jobs in each event is called an event scheduling problem, for which a Stochastic Multi-stage job Scheduling (SMS) component is proposed. The event scheduling problem can be regarded as a generalized flowshop. However, it is different from the traditional one in two fundamental aspects: (i) Some VMs are non-available at the time when an event occurs because some jobs or tasks in previous events might still be in processing at that moment. (ii) VMs are not fixed during the whole scheduling process. Since some jobs in the previous event(s) might not have finished when a new event E_q ($q = 2, \ldots, Q$) arrives, we propose two alternations (Job Collecting Strategy, JCS for short) to determine the jobs to be scheduled: keeping the jobs scheduled in the previous event(s) unchanged, or incorporating not started jobs in the previous event(s) into E_q.

The DES (Dynamic Event Scheduling) framework for scheduling dynamic events is detailed in Algorithm 1. The key idea of DES is to collect and arrange jobs periodically.

Algorithm 1: Dynamic Event Scheduling framework (DES)

```
1 Initialization;
2 S ← SMS(E_1);/* Calling Stochastic Multi-stage job Scheduling. */
3 for q = 2 to Q do
4 |   JCS(E_q);/* Calling Job Collecting Strategy */
5 |   S ← S ⋃ SMS(E_q);
6 return S.
```

As can be seen, DES relies on two main operators: the Stochastic Multi-stage job Scheduling (SMS, Line 2) and the Job Collecting Strategy (JCS, Line 4). The details of the SMS and JCS procedures are introduced in the following sections.

4.1 Stochastic Multi-stage job Scheduling (SMS)

In the event scheduling problem, $n_q \times m$ tasks in E_q ($q = 1, \ldots, Q$) and their dependencies are represented as a Directed Acyclic Graph (DAG). Tasks are assigned to VMs in the topological order of the DAG. A task is ready to be assigned only if its immediate predecessors have been assigned to a VM. Since tasks in one job have a linear processing route, at most n_q tasks are ready to be assigned at any moment. Therefore, there are $(n_q!) \times n_q^{n_q(m-1)}$ possible topological sequences for each E_q.

Since each job has at most one ready task at any time and the ready task processing order changes dynamically, we adopt minimum heaps to maintain the corresponding job sequence ζ. i.e., the top task in the heap is processed first. Initially, ζ contains only the first tasks (those of the n_q jobs on Stage 1). ζ is updated by a given rule (two rules with different task removing and appending strategies will be introduced later).

We propose an iterated heuristic framework SMS (Stochastic Multi-stage job Scheduling) for task scheduling in each E_q, which is detailed in Algorithm 2. The key idea of SMS is to take the task scheduling problem in an event as an static optimization problem and solve it by a local search procedure. An initial job sequence is usually constructed by a simple rule, which is evaluated by Timetabling (Line 1). Timetabling decides the way to produce the topological sequence based on the given job sequence, and arranges tasks in order of the topological sequence (Line 2). The job sequence is improved by Sequencing (Line 4) followed by Timetabling to evaluate the quality of the improved sequence (Line 5). The improvement operation is repeated until a termination condition is met (Line 3). There are two termination conditions: one is that no improvement is made by Sequencing and the other one is that the computation time of SMS exceeds a predefined upper bound. Jobs in each event should be scheduled within t, i.e., the computation time of SMS should be limited otherwise the obtained sub-schedule is invalid. We use $T_{\max}$ to denote the upper bound of the computation time of SMS. Therefore, $T_{\max} \leq t$.

Algorithm 2: Stochastic Multi-stage job Scheduling SMS(E_q)

```
1 Initial Job Sequence Construction for E_q;
2 Evaluate the initial job sequence by Timetabling;
3 while (termination conditions not met) do
4 |   Improve the job sequence by Sequencing;
5 |   Evaluate the improved job sequence by Timetabling;
6 return obtained sub-schedule.
```

4.1.1 Initial Job Sequence Construction

In this paper, the Earliest Due Date (EDD) rule [60] is adopted to construct an initial job sequence $\pi^q = (\pi^q_{[1]}, \ldots, \pi^q_{[n_q]})$ in which $\pi^q_{[j]} \in \mathbb{J}_q$ is the j^{th} element of π^q for the coming E_q. The EDD rule requires that $D_{[j_1]} < D_{[j_2]}$ for any $\pi^q_{[j_1]}$ and $\pi^q_{[j_2]}$ with $j_1 < j_2$.

4.1.2 Timetabling Methods

Since the traditional timetabling problem for 2-stage multi-machine flowshops [61] is already NP-hard, it follows that the involved timetabling problem with dynamic workload requirements is also NP-hard which drives us develop effective and efficient heuristics for Timetabling. In this paper, two timetabling methods, STM (Stage-pass Timetabling Method) and TTM (Task-pass Timetabling Method), are presented for allocating tasks in E_q arriving at time α_q to idle time slots according to a given job sequence π^q.

After some tasks are allocated to VMs, some available horizontal time slots might be generated on each VM. $\mathbb{ST}^o_i$ maintains idle time slots of on-premise VMs in V_i and $\mathbb{ST}^r_i$ maintains those of rented VMs in V_i. $< v^k_i, [\mathfrak{a}, \mathfrak{b}] >$ is used to represent the idle time slot $[\mathfrak{a}, \mathfrak{b}]$ on v^k_i ($i = 1, \ldots, m, k = 1, \ldots, o_i + r_i$). $\mathbb{ST}^o_i$ and $\mathbb{ST}^r_i$ are maintained by balanced search trees for a fast search where any node on each tree represents idle time slots with the same start time and the key of the node is the start time of the time slots. For simplicity, n_q dummy tasks $\mathcal{T}_{j,m+1}$ are added with earliest feasible start times $b_{j,m+1} = +\infty$. Considering the computation time of SMS, arrival times of dummy tasks $\mathcal{T}_{j,0}$ are set to $\alpha_q + T_{\max}$.

Either STM or TTM allocate every task $\mathcal{T}_{j,i}$ removed from ζ to a VM using the proposed ARRANGE procedure. The ARRANGE procedure is formally described as Algorithm 3. The key idea of ARRANGE is to arrange the task as early as

possible without violating any constraints, meanwhile the renting cost is minimal.

Firstly ARRANGE tries to find the first feasible idle time slot with $[\mathfrak{a}, \mathfrak{b}]$ satisfying $t_j \leq \max\{c_{j,i-1}, \mathfrak{a}\} + d_{j,i} \leq \min\{C_{j,i}, \mathfrak{b}\} - p_{j,i}$ on $\mathbb{ST}_i^o$ (Line 1) and obtains the start and completion times of $\mathcal{T}_{j,i}$ (Line 3). $\mathbb{ST}_i^o$ is updated after the allocation (Line 4-9).

Fig. 3 gives an example for steps in Lines 1-9 in Algorithm 3. Fig.3 (a) shows the Gantt chart on VC_i (shaded blocks are occupied time slots) with the corresponding $\mathbb{ST}_i^o$ shown in Fig.3 (b). ARRANGE searches on $\mathbb{ST}_i^o$ and arranges $\mathcal{T}_{j,i}$ with $c_{j,i-1} = 30$, $p_{j,i} = 10$, $d_{j,i} = 0$ and $C_{j,i} = 50$. Since the first feasible idle time slot is $< 4, [0, 40] >$, ARRANGE assigns $\mathcal{T}_{j,i}$ to the time slot $< 4, [30, 40] >$ as depicted in Fig.3 (c) and updates $\mathbb{ST}_i^o$ as illustrated in Fig.3 (d).

If no such time slot exists, ARRANGE searches on $\mathbb{ST}_i^r$ for the first feasible idle time slot with $[\mathfrak{a}, \mathfrak{b}]$ satisfying $t_j \leq \max\{c_{j,i-1}, \mathfrak{a}\} + d_{j,i} \leq \min\{C_{j,i}, \mathfrak{b}\} - p_{j,i}$ (Line 11) and obtains the start and completion times of $\mathcal{T}_{j,i}$ (Line 13) . $\mathbb{ST}_i^r$ is updated after the allocation (Line 14-19). If all time slots on both on-premise and rented VMs cannot meet the requirement a new VM is rented to which $\mathcal{T}_{j,i}$ is allocated (Line 21-24). Both $\mathbb{ST}_i^o$ and $\mathbb{ST}_i^r$ are updated after the allocation (Line 25-26).

Algorithm 3: ARRANGE($\mathcal{T}_{j,i}$)

1 Search $\mathbb{ST}_i^o$ for the first feasible idle time slot $[\mathfrak{a}, \mathfrak{b}]$ with $t_j \leq \max\{c_{j,i-1} + d_{j,i-1}, \mathfrak{a}\} \leq \min\{C_{j,i}, \mathfrak{b}\} - p_{j,i}$;
2 **if** $< v_i^k, [\mathfrak{a}, \mathfrak{b}] >$ *is found* **then**
3 $b_{j,i} \leftarrow \max\{c_{j,i-1} + d_{j,i-1}, \mathfrak{a}\}$; $c_{j,i} \leftarrow b_{j,i} + p_{j,i}$;
4 Remove $< v_i^k, [\mathfrak{a}, \mathfrak{b}] >$ from $\mathbb{ST}_i^o$;
5 **if** $b_{j,i} > \mathfrak{a}$ **then**
6 Add $< v_i^k, [\mathfrak{a}, b_{j,i}] >$ to $\mathbb{ST}_i^o$;
7 **if** $c_{j,i} < \mathfrak{b}$ **then**
8 Add $< v_i^k, [c_{j,i}, \mathfrak{b}] >$ to $\mathbb{ST}_i^o$;
9 $v_{j,i} \leftarrow v_i^k$;
10 **else**
11 Search $\mathbb{ST}_i^r$ for the first feasible idle time slot $[\mathfrak{a}, \mathfrak{b}]$ with $t_j \leq \max\{c_{j,i-1} + d_{j,i-1}, \mathfrak{a}\} \leq \min\{C_{j,i}, \mathfrak{b}\} - p_{j,i}$;
12 **if** $< v_i^k, [\mathfrak{a}, \mathfrak{b}] >$ *is found* **then**
13 $b_{j,i} \leftarrow \max\{c_{j,i-1} + d_{j,i-1}, \mathfrak{a}\}$; $c_{j,i} \leftarrow b_{j,i} + p_{j,i}$;
14 Remove $< v_i^k, [\mathfrak{a}, \mathfrak{b}] >$ from $\mathbb{ST}_i^r$;
15 **if** $b_{j,i} > \mathfrak{a}$ **then**
16 Add $< v_i^k, [\mathfrak{a}, b_{j,i}] >$ to $\mathbb{ST}_i^r$;
17 **if** $c_{j,i} < \mathfrak{b}$ **then**
18 Add $< v_i^k, [c_{j,i}, \mathfrak{b}] >$ to $\mathbb{ST}_i^r$;
19 $v_{j,i} \leftarrow v_i^k$;
20 **else**
21 $r_i \leftarrow r_i + 1$;
22 Rent a new VM $v_i^{r_i}$;
23 $b_{j,i} \leftarrow c_{j,i-1} + d_{j,i-1}$;
24 $v_{j,i} \leftarrow v_i^{r_i}$;
25 Add $< v_i^{r_i}, [0, b_{j,i}] >$ to $\mathbb{ST}_i^r$;
26 Add $< v_i^{r_i}, [c_{j,i}, +\infty] >$ to $\mathbb{ST}_i^r$;
27 **return** $< \mathcal{T}_{j,i}, b_{j,i}, v_{j,i} >$.

The time complexity of ARRANGE is determined by the time complexity of searching for the first feasible time slot for the task, which depends on the number of nodes on $\mathbb{ST}_i^o$ and $\mathbb{ST}_i^r$. Suppose n_i^k jobs have been assigned to v_i^k after α_q which result in at most $n_i^k + 1$ idle time intervals on v_i^k. In other words, there are at most $\sum_{k=1}^{o_i}(n_i^k + 1)$ nodes on $\mathbb{ST}_i^o$ and at most $\sum_{k=o_i+1}^{o_i+r_i}(n_i^k + 1)$ idle time slots on $\mathbb{ST}_i^r$. $\sum_{k=1}^{o_i} n_i^k$ is actually the total number of tasks assigned to on-premise VMs after α_q, and $\sum_{k=o_i+1}^{o_i+r_i} n_i^k$ is actually the total number of tasks assigned to rented VMs after α_q. However, the number of currently scheduled tasks is not more than $|S|$, i.e., the worst time complexity of ARRANGE is $O(\log |S|)$.

The Stage-pass Timetabling Method (STM) generates the timetable of tasks in E_q stage by stage (Algorithm 4). Given the job sequence $\pi^q = (\pi^q_{[1]}, \ldots, \pi^q_{[n_q]})$, the ready task sequence $\zeta = (\zeta_{[1]}, \ldots, \zeta_{[n_q]})$ is initialized by setting $\zeta_{[j]} = \mathcal{T}_{\pi^q_{[j]},1}$, i.e., putting Stage 1 tasks into ζ in the order of the given job sequence (Line 2-3). Tasks in ζ are sequentially allocated to VMs by ARRANGE (Line 6-8). Once a task of Stage i ($i = 1, \ldots, m$) in ζ is arranged, it is replaced by its immediate successive task (Line 9). ζ is updated by sequencing tasks in non-decreasing order of their earliest feasible start times, i.e., the completion time of their immediate predecessor tasks (Line 10).

Fig.4 depicts a detailed example for the STM procedure given $\pi^q = (3, 1, 2)$. ζ is expressed in form of a queue. Initially, there are three Stage 1 tasks in ζ: $\mathcal{T}_{3,1}$, $\mathcal{T}_{1,1}$ and $\mathcal{T}_{2,1}$ corresponding to π^q. Their earliest feasible start times are 10. The ARRANGE procedure is called sequentially for $\mathcal{T}_{3,1}$, $\mathcal{T}_{1,1}$ and $\mathcal{T}_{2,1}$. After ARRANGE($\mathcal{T}_{3,1}$), $\mathcal{T}_{3,1}$ is replaced by $\mathcal{T}_{3,2}$ and its earliest feasible start time is 50. After ARRANGE($\mathcal{T}_{1,1}$), $\mathcal{T}_{1,1}$ is replaced by $\mathcal{T}_{1,2}$ and its earliest feasible start time is 25. After ARRANGE($\mathcal{T}_{2,1}$), $\mathcal{T}_{2,1}$ is replaced by $\mathcal{T}_{2,2}$ and its earliest feasible start time is 30. Then tasks in ζ are updated to $\mathcal{T}_{1,2}$, $\mathcal{T}_{2,2}$ and $\mathcal{T}_{3,2}$ corresponding to their earliest feasible start times. The above processes is repeated until all tasks are scheduled.

The time complexity of ζ updating is $O(n_q \log n_q)$. Since ζ is updated m times and ARRANGE is only called once for each task, the time complexity of STM is $O(mn_q \log n_q + mn_q \log |S|)$.

Algorithm 4: Stage-pass Timetabling Method STM(π^q)

1 $s_q \leftarrow \varnothing$;
2 **for** $j = 1$ **to** n_q **do**
3 $\zeta_{[j]} = \mathcal{T}_{\pi^q_{[j]},1}$;
4 **for** $i = 1$ **to** m **do**
5 **for** $j = 1$ **to** n_q **do**
6 $\mathcal{I}_j \leftarrow \arg\{\zeta_{[j]}\}$/* $\mathcal{I}_j$ is the job index of $\zeta_{[j]}$ */
7 $< \zeta_{[j]}, b_{\mathcal{I}_j,i}, v_i^k > \leftarrow$ ARRANGE($\zeta_{[j]}$);
8 $s_q \leftarrow s_q \bigcup \{< \zeta_{[j]}, b_{\mathcal{I}_j,i}, v_{\mathcal{I}_j,i} >\}$;
9 $\zeta_{[j]} \leftarrow \mathcal{T}_{\mathcal{I}_j,i+1}$;
10 Sort tasks in ζ in the non-decreasing order of their earliest feasible start times;
11 **return** s_q.

STM is inspired from the general ideas for solving timetabling problems in the flexible flowshop scheduling [62], in which the number of machines are fixed. Allocations produced by STM follow a stage by stage strategy, i.e., a Stage i task can be scheduled only when all tasks of Stage $i-1$ have been scheduled. Actually, according to the precedence constraint, a task is ready to be scheduled once after its previous task has been scheduled. Generating a timetable of the tasks in a task by task way might produce a different timetable from the one obtained by STM. Therefore, we propose the Task-pass Timetabling Method (TTM, Algorithm 5). Given the job sequence $\pi^q = (\pi^q_{[1]}, \ldots, \pi^q_{[n_q]})$, the ready task sequence $\zeta = (\zeta_{[1]}, \ldots, \zeta_{[n_q]})$ is initialized by setting $\zeta_{[j]} = \mathcal{T}_{\pi^q_{[j]},1}$, i.e., putting Stage 1 tasks into ζ in the order of the given job sequence

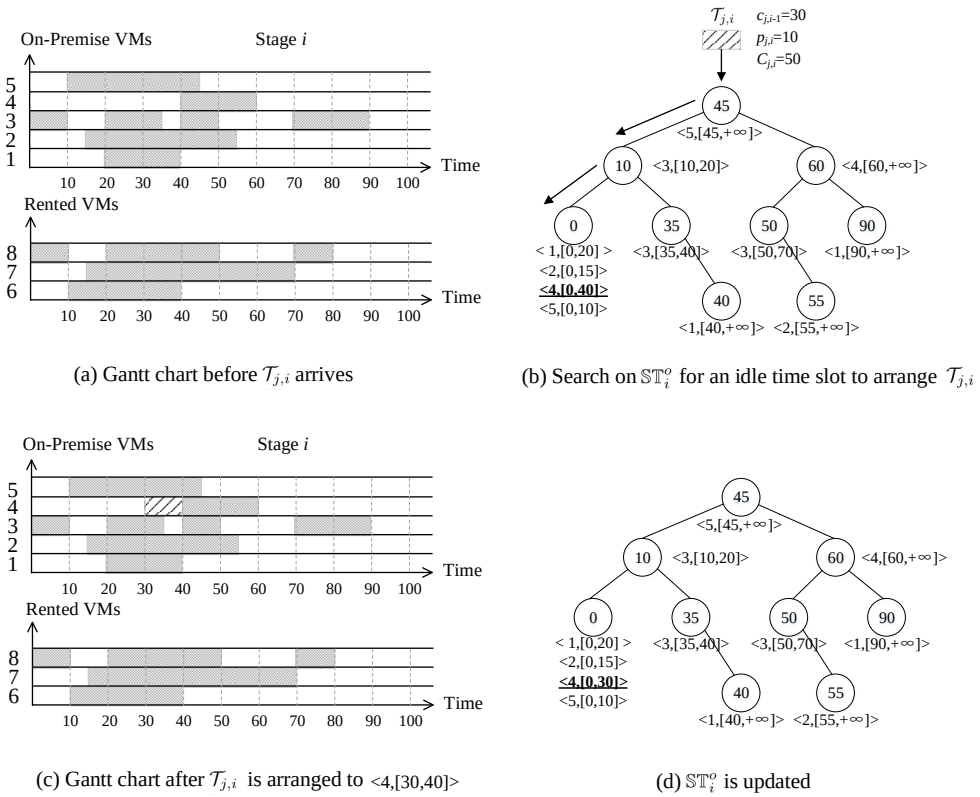


Fig. 3. Example of the ARRANGE procedure (Algorithm 3) for a task.

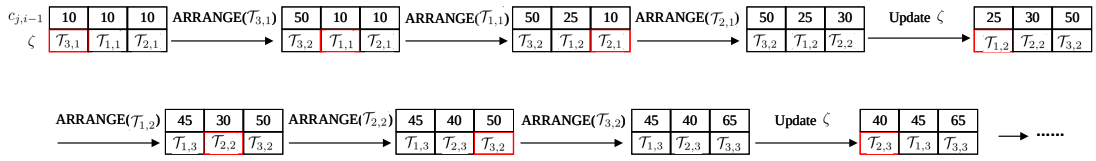


Fig. 4. An example of STM (Algorithm 4) with $\pi^q = (3, 1, 2)$.

(Line 2-3). $\zeta_{[1]}$ is removed and allocated (Line 5-7). The immediate successive task of $\zeta_{[1]}$ is added to ζ (Line 8). ζ is adjusted immediately by sequencing tasks in non-decreasing order of their earliest feasible start times (Line 9). The above procedure is repeated until all tasks are allocated by ARRANGE.

Fig.5 shows a detailed example of TTM given $\pi^q = (3, 1, 2)$. ζ is expressed in the form of a binary tree and the minimal heap adjustment is used for updating ζ. Initially, the top task $\zeta_{[1]}$ is $\mathcal{T}_{3,1}$ and its earliest feasible start time is 10. After ARRANGE($\mathcal{T}_{3,1}$), $\zeta_{[1]}$ is replaced by $\mathcal{T}_{3,2}$ and its earliest feasible start time is 50. ζ is updated immediately by the minimal heap adjustment. After the adjustment, the top task $\zeta_{[1]}$ becomes $\mathcal{T}_{1,1}$. The above procedure is repeated until all tasks are scheduled.

ζ is updated $m \times n_q$ times and the time complexity for updating ζ is only $O(\log n_q)$ by using the minimal heap adjustment. ARRANGE is called once for each task. Therefore, the time complexity of STM is $O(mn_q \log n_q + mn_q \log |S|)$.

Algorithm 5: Task-pass Timetabling Method TTM(π^q)

```
1  s_q ← ∅;
2  for j = 1 to n_q do
3      ζ_[j] = T_{π^q_[j],1};
4  for k = 1 to m × n_q do
5      I_1 ← arg{ζ_[1]} /* I_1 is the job index of ζ_[1]   */
6      < ζ_[1], b_{I_1,i}, v_i^k > ← ARRANGE(ζ_[1]);
7      s_q ← s_q ⋃ {< ζ_[1], b_{I_1,i}, v_{I_1,i} >};
8      ζ_[1] ← T_{I_1,i+1};
9      Sort tasks in ζ in the non-decreasing order of their earliest feasible
         start times;
10 return s_q.
```

4.1.3 Variable Neighborhood Descent Methods for Sequencing

Effective methods are desirable for the NP-hard sequencing problem. In this paper, a variable neighborhood descent (VND) method [63] is proposed to improve job sequences. Two general neighborhood structures are employed: One-Insertion $\mathcal{N}_1$ and Pair-wise Exchange $\mathcal{N}_2$ [64]. $\mathcal{N}_1$ chooses two different jobs from the job sequence and the second job is inserted just before the first one. $\mathcal{N}_2$ chooses two different jobs from the job sequence and swaps them.

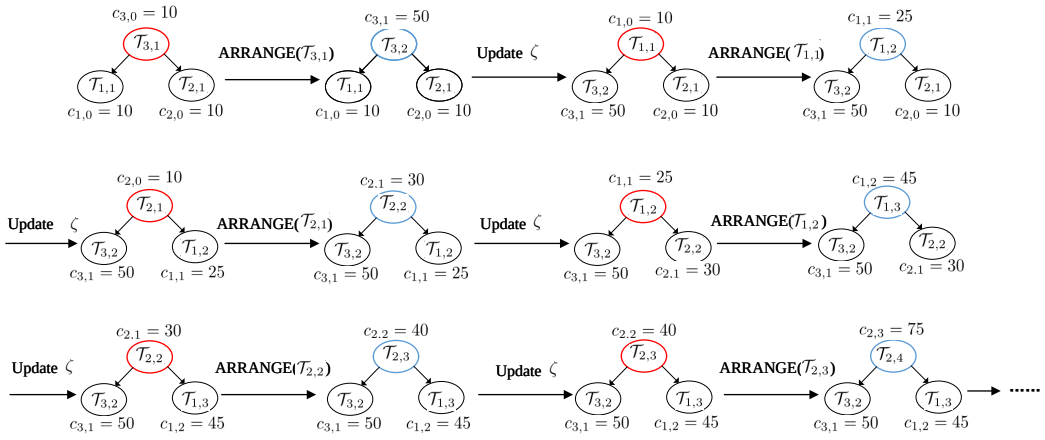


Fig. 5. An example of TTM (Algorithm 5) with $\pi^q = (3, 1, 2)$.

The proposed VND employs greedy local search processes. Generally VND stops when no better solution can be found. According to the above analysis, the termination criterion is set as either no better solution can be found or its execution time reaches $T_{\max}$.

VND is formally described in Algorithm 6. In each iteration (Line 4-15), there are three major steps. Firstly, the neighborhood Π is generated by $\mathcal{N}_k$ ($k = 1, 2$) (Line 4). Secondly, all neighbors in Π are evaluated by the specified metric $F \in \{F_1, F_2, F_3\}$ and the current best solution is updated accordingly (Line 5-11). Thirdly, the neighborhood structure is changed if no better solution is found in the iteration (Line 12-15). In each iteration, at most $n_q \times (n_q - 1)$ neighbors are evaluated by Timetabling, i.e., the time complexity of each iteration of VND is $O(mn_q^3 \log n_q + mn_q^3 \log |S|)$. The time complexity of VND is difficult to determine because the number of iterations performed in the VND loop cannot be expressed in a closed form.

Algorithm 6: Variable Neighborhood Descent VND(π_q,s_q)

```
1  T_start ← T_current /* Record the current time */
2  k ← 1; flag ← FALSE;
3  while k ≤ 2 do
4      Π ← N_k(π_q) /* Generate neighborhood by N_k */
5      for π'_q ∈ Π do
6          s'_q ← Timetabling(π'_q);
           /* The sub-schedule is evaluated by the metric
              F ∈ {F_1, F_2, F_3} */
7          if F(S ∪ s'_q) < F(S ∪ s_q) then
8              π_q ← π'_q; s_q ← s'_q;
9              flag ← TRUE;
10         if T_current − T_start ≥ T_max then
11             return s_q;
12     if flag then
13         k ← 1;
14     else
15         k ← k + 1;
16 return s_q;
```

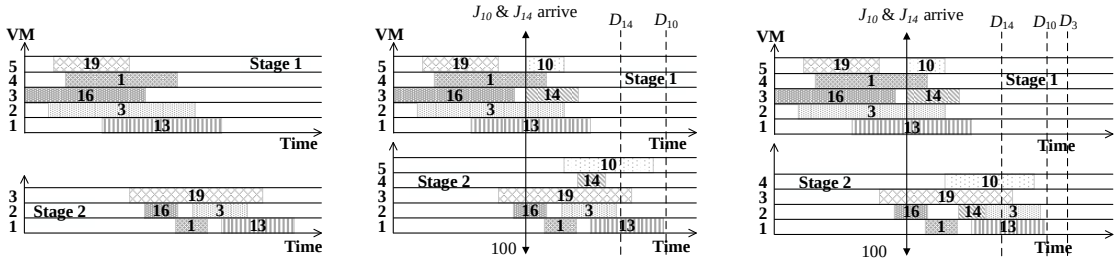
4.2 Job Collecting Strategies

A general strategy for job collection is to only collect new tasks arriving in the last time interval t. This is intuitive and easy to implement. However, this may result in a low solution quality. For example, Fig. 6 (a) shows a Gantt chart before JRE $E_q =< 100, \{J_{10}, J_{14}\} >$. At time point 100, E_q arrives. Two new VMs are rented for new jobs in order to comply with the deadline constraint as shown in Fig. 6 (b). The sub-schedule for E_q does not affect the sub-schedules of previous events. At the time point 100, $\mathcal{T}_{3,2}$ and $\mathcal{T}_{13,2}$ have not yet started. Fig.6 (c) shows the Gantt chart in which $\mathcal{T}_{3,2}$ is re-scheduled. The solution only needs one new VM to accommodate all tasks, thus it is better than the sub-schedule in Fig. 6 (b). In other words, collecting assigned and not started tasks from previous events besides new tasks may lead to better solutions.

Two Job Collecting Strategies are proposed for the general 1. JCS1 only collects tasks of jobs arriving in the last time interval t. Sub-schedules generated by SMS for these jobs make no change to sub-schedules of previous events. However, JCS2 collects some old and not started tasks from previous events along with new tasks. SMS will re-schedule these old tasks.

At each time interval when a new event arrives, JCS2 (Algorithm 7) re-schedules not started tasks from previous events. Dependent tasks to be rescheduled are taken as a new artificial job. For example, at the time point when E_q arrives, tasks from $\mathcal{T}_{j,m'}$ to $\mathcal{T}_{j,m}$ of J_j in a previous sub-schedule have not yet started, i.e., $b_{j,i} > \alpha_q, i = m', m' + 1, \ldots, m$ (Line 4). We construct a new artificial job $J_{j'}$ with $m' - 1$ dummy tasks (Line 6-7) and $m - m' + 1$ ordinary tasks (Line 8-10). For each dummy task, the starting time $b'_{j,i}$ ($i = 1, \ldots, m' - 1$) is set to $\alpha_q + T_{\max}$ and the processing time $p'_{j,i}$ ($i = 1, \ldots, m' - 1$) is set to zero. Ordinary tasks are tasks of J_j that have not yet begun at $\alpha_q + T_{\max}$ and will be rescheduled together with tasks from E_q. An ordinary task $\mathcal{T}_{j',i}$ is identical to $\mathcal{T}_{j,i}$ ($m' \leq i \leq m$).

In addition, heavy workloads usually result in a lot of artificial jobs which would further lead to the computation time of SMS exceeding $T_{\max}$. Therefore, the number of artificial jobs should be limited. In this paper, we limit the number of artificial jobs to ϖ (Line 2), which are selected from the most recent events (Line 3).



(a) Gantt chart before J_{10} & J_{14} arrive

(b) Gantt chart obtained by DES with JCS1 collecting only new tasks

(c) Gantt chart obtained by DES with JCS2 collecting some old tasks

Fig. 6. Gantt charts obtained by DES

Algorithm 7: Job Collecting Strategy 2 JCS2(E_q)

Input: ϖ
$count \leftarrow 0$;
while $count \leq \varpi$ **do**
  $j \leftarrow \arg\max\{b_{j,i} | < \mathcal{T}_{j,m}, b_{j,m}, v_{j,m} > \in S\}$;/* Find the job index of the assigned task that has the latest beginning time */
  Find $m' \leq m$ satisfying $b_{j,m'} \geq \alpha_q + T_{\max}$ and $b_{j,m'-1} < \alpha_q + T_{\max}$;
  if *(m' exists)* **then**
    for $i = 1$ **to** $m' - 1$ **do**
      Generate dummy tasks $\mathcal{T'}_{j,i}$ with $b'_{j,i} = \alpha_q + T_{\max}$ and $p'_{j,i} = 0$;
    for $i = m'$ **to** m **do**
      Generate ordinary tasks $\mathcal{T'}_{j,i}$ with $p'_{j,i} = p_{j,i}$;
      $S \leftarrow S - \{< \mathcal{T}_{j,i}, b_{j,i}, v_{j,i} >\}$/* Remove the assignment of $\mathcal{T}_{j,i}$ */
    Generate the artificial job J'_j consisting of $\mathcal{T'}_{j,1}, \ldots, \mathcal{T'}_{j,m}$ with a linear processing route;
    $\mathbb{J}_q \leftarrow \mathbb{J}_q \cup \{J'_j\}$;
    $count \leftarrow count + 1$;
  else
    break;
return

5 Computational Experiments

There are several variants for each component or parameter of the proposed DES framework. We calibrate parameters and components first based on which three heuristics are compared. All tested algorithms are coded in Java and run on an Intel Core $i7-4790$ CPU @$3.60GHz$ with 8 GBytes of RAM.

5.1 Parameter and Component Calibration

Since there are no comprehensive benchmarks available for the dynamic problem under study, we generate testing instances based on studies for the workload allocation in cloud services [18], [42]. We consider the workload within one hour, i.e. 3600 JREs ($Q = 3600$) or one JRE per second ($t = 1000ms$). Two probability distributions are explored for generating the number of jobs in JRE E_q, i.e. n_q: (1) Poisson distribution $P(\lambda)$, $\lambda \in \{5, 10, \ldots, 50\}$ and (2) Uniform distribution $U(5, 50)$. The deadline of J_j is constructed by $D_j = t_j + (df_j + 1) \times \sum_{i=1}^{m} p_{j,i}$ in which df_j is the deadline factor obeying the normal distribution $N(df_{avg}, \sigma^2)$, $df_{avg} \in \{0.5, 1, 1.5, 2\}$, $\sigma = 0.1 \times df_{avg}$. The processing time of a task can be approximately taken as a function of the data size to be processed according to [18], in which the well-known word-count and word-median programs are tested on VMs provided by Amazon EC2. There is no general distribution pattern for the size of data that a user requests to process. In order to make the instances more reasonable, the processing time of a task is set randomly between $[1, 10]$ seconds. Note that the processing times should not be too small (less than 1 second), otherwise the network latency could seriously affect the response times of the application, then deploying the application across multiple clusters will be meaningless. Since the application considered in the paper is not large-scale application, e.g., the image processing application, the processing time of a task is not set to a large value (less than 10 seconds). We assume workloads on all virtual clusters are the same on average, then o_i is initialized to be the same for each virtual cluster. At the same time, there would be a high probability of the ceiling effect taking place if o_i was set to a large value, i.e., it is easy to obtain optimal solutions (in which there is no rented VM), even by a roughly designed algorithm, because on-premise VMs were sufficient. Therefore, we employ Eq. (11) and Eq. (12) to determine the value of o_i for the instances with $n_q \sim P(\lambda)$ and $n_q \sim U(5, 50)$, respectively.

$$o_i = \frac{\lambda \times \sqrt{m}}{df_{avg}} \tag{11}$$

$$o_i = \frac{25 \times \sqrt{m}}{df_{avg}} \tag{12}$$

The network bandwidth varies with time. In [42], the weighted average network delay over time to the three remote clouds are measured. Based on the measurements in [42], the network delay $d_{j,i}$ obeys the uniform distribution on a given interval: $[10ms, 15ms]$, $[70ms, 80ms]$ or $[270ms, 280ms]$.

The involved instance parameters are set as $m \in \{3, 5, 10\}$, $df_{avg} \in \{0.5, 1, 1.5, 2\}$ and $d_{j,i} \in [10ms, 15ms]$, $[70ms, 80ms]$ or $[270ms, 280ms]$. For $n_q \sim P(\lambda)$ ($\lambda \in \{5, 10, \ldots, 50\}$), there are $3 \times 10 \times 4 \times 3 = 360$ instance combinations, and 10 calibration instances are randomly generated for each one of these combinations. For $n_q \sim$

$U(5, 50)$, there are $3 \times 4 \times 3 = 36$ instance combinations, and 100 calibration instances are randomly generated also for each possible combination. Therefore, $360 \times 10 + 36 \times 100 = 7200$ instances in total are tested to calibrate the parameters and components. Solutions are evaluated by RPD (Relative Percentage Deviation) defined as follows:

$$RPD(\%) = \frac{F - F_{Best}}{F_{Best}} \times 100\% \quad (13)$$

where F is the objective value obtained by the corresponding algorithm and F_{Best} is the best objective value obtained.

In the DES framework, there are two variants for the timetabling component (STM and TTM), the sequencing component (VND or None) and the JCS strategy (JCS1 or JCS2) respectively, and three metrics to optimize (F_1, F_2 and F_3). In other words, there are $2 \times 2 \times 2 \times 3 = 24$ component combinations. However, $T_{\max}$ restricts the running time of VND. We test 10 cases of $T_{\max} \in \{100, 200, 300, \ldots, 1000\}$ milliseconds, i.e., 10 variants of VND are tested. Along with the None case, there are 11 variants in total for the sequencing component. In addition, ϖ limits the number of artificial jobs to re-schedule in each JRE of JCS2. If ϖ is too large, timetabling will spend too much time on generating a schedule, and fewer candidates will be explored within $T_{\max}$ which leads to low quality solutions for the local search. We test all values of $\frac{\varpi}{n_q} \in \{0.1, 0.2 \ldots, 1\}$, i.e., 10 variants of JCS2 are tested. Considering the 1 JCS1 variant, there are 11 variants for the JCS component. Therefore, the total number of component and parameter combinations is $2 \times 11 \times 11 \times 3 = 726$, i.e., $726 \times 7200 = 5,227,200$ experimental results are obtained.

Experimental results are analyzed by the multi-factor analysis of variance (ANOVA) technique. First, the three main hypotheses (normality, homoscedasticity, and independence of the residuals) are checked from the residuals of the experiments. All three hypotheses are acceptable from this analysis. Since all the p-values in the experiments are close to zero, they are not given in this paper. Greater F-Ratios imply factors with stronger effects. Interactions between (or among) any two (or more than two) factors are not considered because the observed F-Ratios are small in comparison.

Fig.7 shows the means plot of $T_{\max}$ factor with different values and 95.0% Tukey HSD intervals. Recall that overlapping intervals indicate statistical insignificance among the overlapped averages. From Fig.7, we can observe that RPD decreases with an increase in $T_{\max}$, i.e., a larger $T_{\max}$ means better solutions are obtained. The differences are statistically significant for $T_{\max} < 800ms$. However, the differences are not statistically significant when $T_{\max} \in \{800, 900, 1000\}ms$. We set $T_{\max} = 900ms$ in the following.

Fig.7 shows the means plot of ϖ/n_q with different values and 95.0% Tukey HSD intervals. From Fig.7, it can be observed that there is an increasing tendency of RPD with the increase of ϖ/n_q, i.e., larger ϖ/n_q always results in worse solutions. The differences are not statistically significant when $\varpi/n_q \in \{0.1, 0.2, 0.3\}$ and it seems that RPD gets the best value when $\varpi/n_q = 0.2$. Therefore, ϖ takes $0.2n_q$ in the following.

Fig. 8-10 shows the means plot of component settings with 95.0% Tukey HSD intervals for the objective F_1, F_2 and $F3$, respectively. It shows a statistically significant difference for the performance of different Timetabling, Sequencing and JCS settings. TTM outperforms STM, VND outperforms not applying VND, and JCS2 performs better than JCS1.

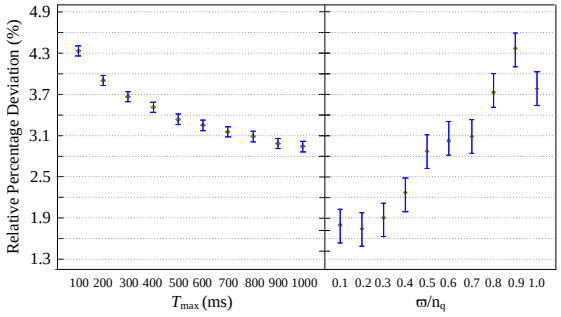


Fig. 7. Means plot of $T_{\max}$ and ϖ/n_q with 95.0% Tukey HSD intervals

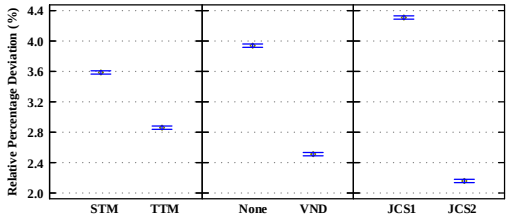


Fig. 8. The mean plot of component settings with 95.0% Tukey HSD intervals for the objective F_1

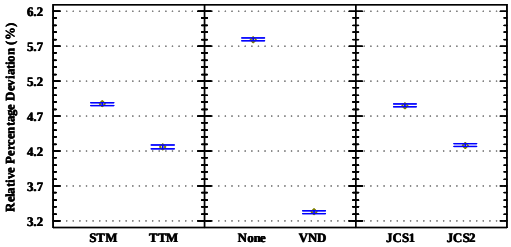


Fig. 9. The mean plot of component settings with 95.0% Tukey HSD intervals for the objective F_2

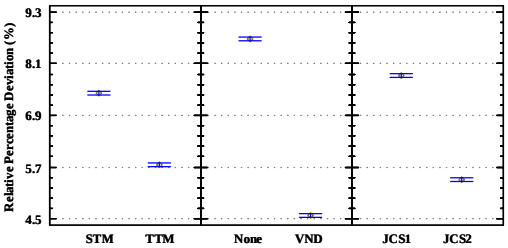


Fig. 10. The mean plot of component settings with 95.0% Tukey HSD intervals for the objective F_3

5.2 Algorithm Comparison

In terms of the above calibration, we set TTM as Timetabling, VND as Sequencing, JCS2 as JCS, 900ms as $T_{\max}$ and $0.2 \times n_q$ as ϖ for the proposed DES.

To the best of our knowledge, there is no existing algorithm for the problem studied in this paper. However, the problem studied in [45] and [53] are related to the problem under study in this paper. In [45], a list based heuristic (HEFT) was proposed for dynamic workflow scheduling, in which jobs are assigned priorities according to their deadlines. In [53], a greedy local search heuristic (LS_ENG) was presented for the static deadline-constraint flowshop scheduling with scalable resources. In order to evaluate the performance of the proposed DES, we adapt HEFT and LS_ENG to the considered problem. The modified HEFT is a special DES with None for Sequencing and JCS1 for JCS. Different from STM and TTM, in which task processing order changes dynamically, the timetabling method used in HEFT arranges tasks in a fixed order. The modified LS_ENG employs the same timetabling method as in HEFT. A greedy local search is adopted for Sequencing and JCS1 for JCS. The computation time of the modified HEFT and LS_ENG for scheduling an event is also limited to 900ms as in DES for a fair comparison.

For each of the 360 instance combinations with $n_q \sim P(\lambda)$, 10 new instances are randomly generated, i.e., in total there are 3600 instances for the three algorithms to compare. For each of the 36 instance combinations with $n_q \sim N(5, 50)$, 100 new instances are randomly generated, i.e., in total there are 3600 instances for the three algorithms to compare. ARPDs (Average Relative Percentage Deviation) on all instance combinations are shown in Table 2-3. Table 2 illustrates that for the problems with $n_q \sim P(\lambda)$, DES has the smallest ARPD on all objectives (2.19%, 3.09% and 3.58% for objectives F_1, F_2 and F_3, respectively) and HEFT has the largest (6.29%, 7.23% and 11.35% for objectives F_1, F_2 and F_3, respectively). Table 3 illustrates that for the problems with $n_q \sim U(5, 50)$, DES has the smallest ARPD on all objectives (1.41%, 3.06% and 3.05% for objectives F_1, F_2 and F_3, respectively) and HEFT has the largest (6.32%, 6.85% and 12.73% for objectives F_1, F_2 and F_3, respectively). The reasons lie in that: (i) Solutions are not improved in the modified HEFT. (ii) DES reschedules some not started tasks while the modified LS_ENG only cares about newly arriving tasks.

To further demonstrate the robustness of the three compared algorithms, we analyze the influence of the three instance parameters on the compared algorithms. Interactions between each parameter and the compared algorithms with 95.0% Tukey HSD intervals are depicted in Fig. 11-Fig. 13 for the objective F_1, F_2 and F_3, respectively.

For the problem with F_1 objective, Fig. 11 shows that the stage number m has a large influence on the performance of LS_ENG but the observed differences in RPD values are not statistically significant for DES and HEFT in all cases. λ has also a big influence on the performance of HEFT. However, RPD differences are not statistically significant for LS_ENG and DES in most cases. df_{avg} has also a large influence on all compared algorithms as the obtained differences in the RPD values are statistically significant on all cases. The interval setting of $d_{j,i}$ has little influence on all compared algorithms. RPD differences are not statistically significant for each algorithm in all cases. For the problem with the objective of F_2, Fig. 12 illustrates that the stage number m does indeed have a great influence on the performance of all

TABLE 2
ARPDs(%) of the compared algorithms on instances with $n_q \sim P(\lambda)$

Prameter	Value	F_1			F_2			F_3		
		HEFT	LS_ENG	DES	HEFT	LS_ENG	DES	HEFT	LS_ENG	DES
m	3	5.95	4.14	2.67	6.30	4.01	3.48	13.83	7.59	5.36
	5	6.47	3.24	1.91	8.05	4.54	3.74	11.90	5.45	2.83
	10	6.34	4.54	2.15	6.99	2.21	2.11	9.03	6.89	3.15
λ	5	4.68	4.67	3.05	5.83	3.96	3.73	9.01	6.77	4.74
	10	5.71	4.15	2.81	7.15	4.26	3.88	9.40	5.96	3.82
	15	7.07	3.90	3.00	8.08	4.60	3.97	11.68	6.43	4.03
	20	7.77	3.32	2.53	7.98	4.34	3.54	12.57	5.98	3.61
	25	7.70	3.37	2.03	7.75	3.91	3.22	12.98	5.84	3.31
	30	7.21	3.54	1.73	7.48	3.42	2.86	12.57	5.73	2.91
	35	6.21	3.64	1.47	7.21	3.05	2.58	11.80	6.08	3.09
	40	5.59	3.99	1.54	7.04	2.67	2.38	11.23	7.05	3.15
	45	5.32	4.34	1.61	6.86	2.49	2.19	11.06	7.63	3.28
	50	5.31	4.63	1.84	6.77	2.45	2.07	11.22	8.19	3.81
df_{avg}	0.5	3.89	2.35	1.11	4.11	3.21	2.48	7.84	4.09	2.71
	1	6.05	3.12	1.63	7.13	4.05	3.52	10.96	4.82	2.54
	1.5	7.08	4.59	2.69	8.50	3.76	3.35	12.69	7.32	4.03
	2	8.06	5.60	3.27	9.08	3.25	3.00	13.79	9.66	4.97
$d_{j,i}$	[10,15]	6.26	4.04	2.23	7.32	3.70	3.20	11.38	6.42	3.43
	[70,80]	6.24	3.94	2.19	7.19	3.54	3.03	11.26	6.45	3.59
	[270,280]	6.38	3.83	2.14	7.18	3.46	3.03	11.42	6.65	3.72
Average		6.29	3.94	2.19	7.23	3.57	3.09	11.35	6.51	3.58

TABLE 3
ARPDs(%) of the compared algorithms on instances with $n_q \sim U(5, 50)$

Prameter	Value	F_1			F_2			F_3		
		HEFT	LS_ENG	DES	HEFT	LS_ENG	DES	HEFT	LS_ENG	DES
m	3	6.04	3.34	1.74	6.53	4.53	3.66	15.59	7.04	4.20
	5	6.74	2.41	1.27	7.50	4.62	3.71	12.51	4.72	2.15
	10	6.26	3.95	1.12	6.64	1.50	1.65	9.14	5.87	2.405
df_{avg}	0.5	3.58	2.05	0.77	3.63	3.18	2.25	7.55	4.16	2.25
	1	5.98	2.54	0.96	6.39	4.11	3.47	11.85	4.05	2.07
	1.5	7.35	3.74	1.61	8.33	3.96	3.48	14.76	6.31	3.07
	2	8.40	4.69	2.32	9.10	3.32	3.07	16.82	9.53	4.83
$d_{j,i}$	[10,15]	6.55	3.12	1.53	7.01	4.03	3.37	13.90	6.09	3.38
	[70,80]	6.20	3.40	1.40	6.64	3.35	2.84	12.34	6.17	3.09
	[270,280]	6.23	3.22	1.33	6.90	3.59	3.02	12.17	5.80	2.78
Average		6.32	3.25	1.41	6.85	3.64	3.06	12.73	6.00	3.05

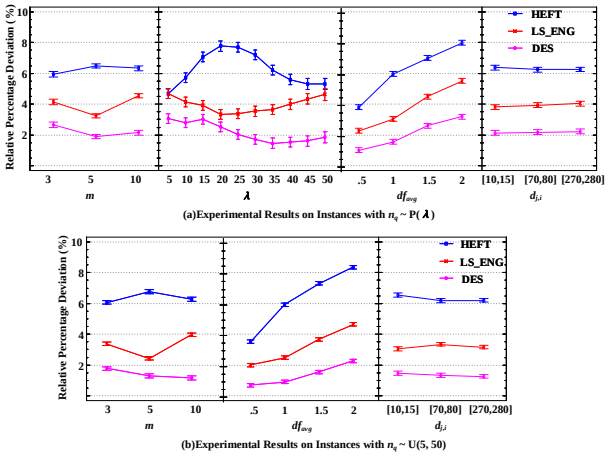


Fig. 11. Interactions between instance parameters and the compared algorithms with 95.0% Tukey HSD intervals for the objective F_1

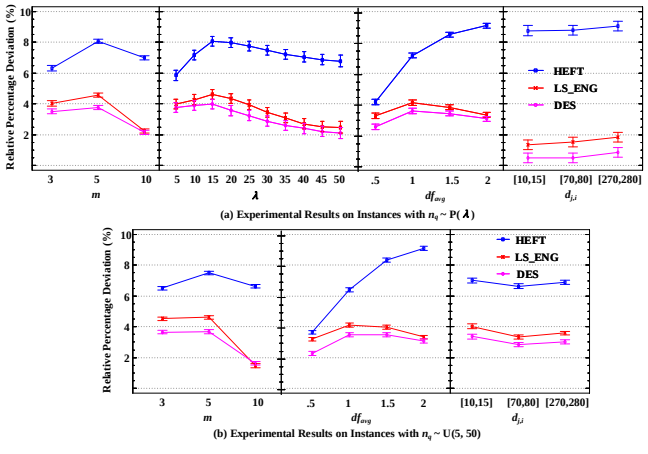


Fig. 12. Interactions between instance parameters and the compared algorithms with 95.0% Tukey HSD intervals for the objective F_2

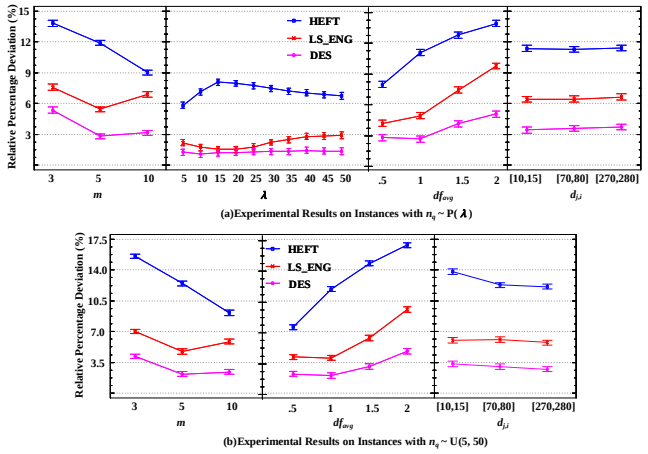


Fig. 13. Interactions between instance parameters and the compared algorithms with 95.0% Tukey HSD intervals for the objective F_3

tested algorithms and the RPD differences are statistically significant between the $m = 10$ case and the remaining cases. λ has a large influence on the performance of HEFT. RPD differences are not statistically significant for LS_ENG and DES in most cases. df_{avg} has a measurable influence on the performance of HEFT. RPD differences are not statistically significant for LS_ENG and DES in all cases. The interval setting of $d_{j,i}$ has little influence on all compared algorithms. RPD differences are not statistically significant for the tested methods. For the problem with the objective of F_3, Fig. 13 shows that the stage number m has a large effect on the performance of HEFT but the RPD values are not statistically different for LS_ENG and DES in most of the cases. Furthermore, λ has a small influence on the tested methods and the RPD values are shown not to be statistically significant. df_{avg} has a great influence on the performance of HEFT and LS_ENG but the RPD values are not statistically different for DES in most of the cases. The interval setting of $d_{j,i}$ has a negligible influence on the RPD values of all compared algorithms.

From the analysis of the results we can conclude that the proposed DES is the most robust among the compared algorithms for all instance parameters (m, λ, df_{avg} and $d_{j,i}$) and all optimized objectives, which implies that DES is suitable for the considered problem.

6 Conclusions

We have considered the stochastic multi-stage job scheduling problem on scalable resources in hybrid clouds to maximize the utilization of rented VMs over a certain period of time. For the problem under study, we proposed the DES framework which consists of a job collecting component (JCS) and a job scheduling component (SMS). JCS periodically collects stochastic jobs and SMS schedules them. In the SMS, we developed two timetabling methods (STM and TTM) for schedule generation and a local search method (VND) for schedule improvement. STM generates the timetable using a stage-by-stage strategy while TTM adopts a task-by-task strategy. When JCS collects some not started tasks for SMS to reschedule, there is a greater probability of DES yielding better solutions. The task-by-task strategy generated better schedules than the stage-by-stage strategy. VND has a positive impact on improving solutions in DES. By comparing with two existing algorithms, we have illustrated that DES statistically outperforms the compared algorithms for the problem under study. In addition, it is not sensitive to instance parameters.

For future research, the problem with non-linear task-dependency is worth studying. Resource provisioning for scenarios with fuzzy processing times and deadlines is also worthy of consideration.

Acknowledgments

This work is sponsored by the National Natural Science Foundations of China (Nos.71401079, 61572127, 61472192), NUPTSF (No. NY214016) and the Collaborative Innovation Center of Wireless Communications Technology. Rubén Ruiz is partially supported by the Spanish Ministry of Economy and Competitiveness, under the project "SCHEYARD – Optimization of Scheduling Problems in Container Yards" (No. DPI2015-65895-R) financed by FEDER funds.

References

[1] X. Zhang, L. T. Yang, C. Liu, and J. Chen, "A scalable two-phase top-down specialization approach for data anonymization using mapreduce on cloud," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 25, no. 2, pp. 363–373, 2014.

[2] X. Li, T. Jiang, and R. Ruiz, "Heuristics for periodical batch job scheduling in a mapreduce computing framework," *Information Sciences*, vol. 326, no. C, pp. 119–133, 2016.

[3] T. Y. Wu, C. Y. Chen, L. S. Kuo, W. T. Lee, and H. C. Chao, "Cloud-based image processing system with priority-based data distribution mechanism," *Computer Communications*, vol. 35, no. 15, pp. 1809–1818, 2012.

[4] M. D. D. Assuncao, C. H. Cardonha, M. A. S. Netto, and R. L. F. Cunha, "Impact of user patience on auto-scaling resource capacity for cloud services," *Future Generation Computer Systems*, vol. 55, pp. 41–50, 2015.

[5] M. Masdari, S. Valikardan, Z. Shahi, and S. I. Azar, "Towards workflow scheduling in cloud computing: A comprehensive analysis," *Journal of Network & Computer Applications*, vol. 66, pp. 64–82, 2016.

[6] W. N. Chen and J. Zhang, "An ant colony optimization approach to a grid workflow scheduling problem with various qos requirements," *IEEE Transactions on Systems Man & Cybernetics Part C*, vol. 39, no. 1, pp. 29–43, 2009.

[7] K. Bochenina, N. Butakov, and A. Boukhanovsky, "Static scheduling of multiple workflows with soft deadlines in non-dedicated heterogeneous environments," *Future Generation Computer Systems*, vol. 55, 2015.

[8] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Algorithms for cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds," *Future Generation Computer Systems*, vol. 48, pp. 1–18, 2015.

[9] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. 2011, pp.1–12.

[10] D. Yun, C. Q. Wu, and Y. Gu, "An integrated approach to workflow mapping and task scheduling for delay minimization in distributed environments," *Journal of Parallel & Distributed Computing*, vol. 84, pp. 51–64, 2015.

[11] E. N. Alkhanak, S. P. Lee, R. Rezaei, and R. M. Parizi, "Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: A review, classifications, and open issues," *Journal of Systems & Software*, vol. 113, pp. 1–26, 2016.

[12] X. Wang, C. S. Yeo, R. Buyya, and J. Su, "Optimizing the makespan and reliability for workflow applications with reputation and a look-ahead genetic algorithm," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1124–1134, 2011.

[13] J.J. Durillo, V. Nae, and R. Prodan, "Multi-objective workflow scheduling: An analysis of the energy efficiency and makespan tradeoff," in *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2013, pp. 203–210.

[14] A. Berl, E. Gelenbe, M.D. Girolamo, G. Giuliani, H.D. Meer, M.Q. Dang and K. Pentikousis, "Energy-efficient cloud computing," in *Computer Journal*, vol. 53, no. 7, pp. 1045–1051, 2010.

[15] E. Gelenbe and S. Timotheou, "Random neural networks with synchronized interactions." *Neural Computation*, vol. 20, no. 9, pp. 2308–2324, 2008.

[16] E. Gelenbe, "Sensible decisions based on QoS," *Computational Management Science*, vol. 1, no. 1, pp. 1–14, 2003.

[17] L. Wang and E. Gelenbe, "Adaptive dispatching of tasks in the cloud," *IEEE Transactions on Cloud Computing*, vol. 99, pp. 1–1, 2015.

[18] Z. Wang, M. M. Hayat, N. Ghani and K. B. Shanban, "Optimizing Cloud-Service Performance: Efficient Resource Provisioning Via Optimal Workload Allocation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 6, pp. 1689–1702, 2017.

[19] I. Al-Azzoni and D. G. Down, "Linear programming-based affinity scheduling of independent tasks on heterogeneous computing systems," *IEEE Transactions on Parallel & Distributed Systems*, vol. 19, no. 19, pp. 1671–1682, 2008.

[20] X. Zuo, G. Zhang, and W. Tan, "Self-adaptive learning pso-based deadline constrained task scheduling for hybrid iaas cloud," *IEEE Transactions on Automation Science & Engineering*, vol. 11, no. 2, pp. 564–573, 2014.

[21] Y. Xu, K. Li, T. T. Khac, and M. Qiu, "A multiple priority queueing genetic algorithm for task scheduling on heterogeneous computing systems," *Information Sciences*, vol. 270, no. 4, pp. 639–646, 2012.

[22] C. Delimitrou and C. Kozyrakis, "QoS-aware scheduling in heterogeneous datacenters with paragon," *Acm Transactions on Computer Systems*, vol. 31, no. 4, pp. 879–889, 2013.

[23] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, D. Brooks, S. Campanoni, K. Brownell, T. M. Jones *et al.*, "Profiling a warehouse-scale computer," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. ACM, 2015, pp. 158–169.

[24] M. Kozlovszky, K. Karoczkai, I. Marton, A. Balasko, A. Marosi, and P. Kacsuk, "Enabling generic distributed computing infrastructure compatibility for workflow management systems," *Computer Science*, vol. 13, no. 3, 2012.

[25] F. Jrad, J. Tao, and A. Streit, "A broker-based framework for multi-cloud workflows," in *International Workshop on Multi-Cloud Applications and Federated Clouds*, 2013, pp. 61–68.

[26] J. Wang, P. Korambath, I. Altintas, J. Davis, and D. Crawl, "Workflow as a service in the cloud: Architecture and scheduling algorithms ," *Procedia Computer Science*, vol. 29, pp. 546–556, 2014.

[27] G. Papuzzo and G. Spezzano, "Autonomic management of workflows on hybrid grid-cloud infrastructure," in *International Conference on Network and Services Management*, 2011, pp. 1–4.

[28] Q. Tao, H. Y. Chang, Y. Yi, C. Q. Gu, and W. J. Li, "A rotary chaotic pso algorithm for trustworthy scheduling of a grid workflow," *Computers & Operations Research*, vol. 38, no. 5, pp. 824–836, 2011.

[29] R. K. Jena, "Multi objective task scheduling in cloud environment using nested pso framework," *Procedia Computer Science*, vol. 57, pp. 1219–1227, 2015.

[30] H. Liu, A. Abraham, V. Snasel, and S. Mcloone, "Swarm scheduling approaches for work-flow applications with security constraints in distributed data-intensive computing environments," *Information Sciences*, vol. 192, no. 6, pp. 228–243, 2013.

[31] L. Cui, J. Zhang, L. Yue, Y. Shi, H. Li, D. Yuan, H. Liu, A. Abraham, V. Snasel, and S. Mcloone, "A Genetic Algorithm Based Data Replica Placement Strategy for Scientific Applications in Clouds," *IEEE Transactions on Services Computing*, vol. PP, no. 99, pp. 1–1, 2015.

[32] S. G. Ahmad, C. S. Liew, E. U. Munir, F. A. Tan, and S. U. Khan, "A hybrid genetic algorithm for optimization of scheduling workflow applications in heterogeneous computing systems," *Journal of Parallel & Distributed Computing*, vol. 87, no. C, pp. 80–90, 2016.

[33] P. Lopez-Garcia, E. Onieva, E. Osaba, A. D. Masegosa, and A. Perallos, "Gace: A meta-heuristic based in the hybridization of genetic algorithms and cross entropy methods for continuous optimization," *Expert Systems with Applications*, vol. 55, pp. 508–519, 2016.

[34] M. H. Eawna, S. H. Mohammed, and E. S. M. El-Horbaty, "Hybrid algorithm for resource provisioning of multi-tier cloud computing ," *Procedia Computer Science*, vol. 65, pp. 682–690, 2015.

[35] S. Kianpisheh, N. M. Charkari, and M. Kargahi, "Reliability-driven scheduling of time/cost-constrained grid workflows," *Future Generation Computer Systems*, vol. 55, pp. 1–16, 2015.

[36] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: distributed, low latency scheduling," in *Twenty-Fourth ACM Symposium on Operating Systems Principles*, 2013, pp. 69–84.

[37] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: flexible, scalable schedulers for large compute clusters," in *ACM European Conference on Computer Systems*, 2013, pp. 351–364.

[38] I. R. A. Rajendran, "Matrix: Many-task computing execution fabric for extreme scales," Department of Computer Science, Illinois Institute of Technology, MS Thesis, 2013.

[39] I. Sadooghi, S. Palur, A. Anthony, I. Kapur, K. Belagodu, P. Purandare, K. Ramamurty, K. Wang, and I. Raicu, "Achieving efficient distributed scheduling with message queues in the cloud for many-task computing and high-performance computing," in *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2014, pp. 404–413.

[40] L. Wang and E. Gelenbe, "Experiments with Smart Workload Allocation to Cloud Servers, " *IEEE, Symposium on Network Cloud Computing and Applications IEEE Computer Society*, pp. 31–35, 2015.

[41] E. Gelenbe and L. Wang, "Tap: A task allocation platform for the EU FP7 PANACEA project, " *Advances in Service-Oriented and Cloud Computing: Workshops of ESOCC*, vol. 567, pp. 425, 2016.

[42] L. Wang, O. Brun and E. Gelenbe, "Adaptive workload distribution for local and remote Clouds, " *IEEE International Conference on Systems, Man, and Cybernetics*, pp. 3984–3988, 2017.

[43] R. Bertin, S. Hunold, A. Legrand, C. Touati, "Fair scheduling of bag-of-tasks applications using distributed lagrangian optimization," in *Journal of Parallel & Distributed Computing*, 2014, vol. 74, no. 1, pp. 1914–1929.

[44] A. Benoit, L. Marchal, J. F.. Pineau, Y. Robert, F. Vivien, "Scheduling concurrent bag-of-tasks applications on heterogeneous platforms," in *IEEE Transactions on Computers*, 2009, vol. 59, no. 2, pp. 202–217.

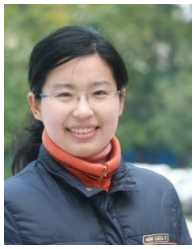
[45] H. Topcuouglu, S. Hariri, and M. Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel & Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[46] M. D. De Assunçao, A. Di Costanzo, and R. Buyya, "Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters," in *Proceedings of the 18th ACM international symposium on High performance distributed computing*. 2009, vol. 13, pp. 141–150.

[47] C.-T. Lu, C.-W. Chang, and J.-S. Li, "Vm scaling based on hurst exponent and markov transition with empirical cloud data," *Journal of Systems and Software*, vol. 99, pp. 199–207, 2015.

[48] L. F. Bittencourt and E. R. M. Madeira, "HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds," *Journal of Internet Services & Applications*, vol. 2, no. 3, pp. 207–227, 2011.

[49] J. Liu, E. Pacitti, P. Valduriez, D. De Oliveira, and M. Mattoso, "Multi-objective scheduling of scientific workflows in multisite clouds," *Future Generation Computer Systems*, vol. 63, no. C, pp. 76–95, 2016.
[50] H.M. Fard, R. Prodan, and T. Fahringer, "A truthful dynamic workflow scheduling mechanism for commercial multicloud environments," *IEEE Transactions on Parallel & Distributed Systems*, vol. 24, no. 6, pp. 1203–1212, 2013.
[51] E. Gelenbe and R. Lent, "Energy-QoS trade-offs in mobile service selection," *Future Internet*, vol. 5, no. 2, pp. 128–139, 2013.
[52] R. Duan, R. Prodan and X. Li, "Multi-objective game theoretic schedulingof bag-of-tasks workflows on hybrid clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 1, pp. 29–42, 2014.
[53] J. Zhu and X. Li, "Scheduling for multi-stage applications with scalable virtual resources in cloud computing," *International Journal of Machine Learning & Cybernetics*, vol. 8, no. 5, pp. 1633–1641, 2017.
[54] J. Zhu and X. Li, "Elastic and flexible multi-stage task scheduling with deadline-constraint in clouds," *The 20th IEEE International Conference on Computer Supported Cooperative Work in Design*, pp. 286–291, 2016.
[55] J. Zhu and X. Li, R. Ruiz, X. Xu, and Y. Zhang, "Scheduling Stochastic Multi-stage Jobs on Elastic Computing Services in Hybrid Clouds," *IEEE International Conference on Web Services*, pp. 678–681, 2016.
[56] [Online]. TOSCA Specification, Available: http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html
[57] G. E. Vieira, J. W. Herrmann, and E. Lin, "Rescheduling manufacturing systems: a framework of strategies, policies, and methods," *Journal of scheduling*, vol. 6, no. 1, pp. 39–62, 2003.
[58] X. Liu,Z. Ni, D. Yuan, Y.. Jiang, Z. Wu, J. Chen,Y. Yang, "A novel statistical time-series pattern based interval forecasting strategy for activity durations in workflow systems," *Journal of Systems & Software*, vol. 84, no. 3, pp. 354–376, 2011.
[59] D. Ouelhadj and S. Petrovic, "A survey of dynamic scheduling in manufacturing systems," *Journal of Scheduling*, vol. 12, no. 4, pp. 417–431, 2009.
[60] H. M. Goldberg, "Analysis of the earliest due date scheduling rule in queueing systems," *Mathematics of Operations Research*, vol. 2, no. 2, pp. 145–154, 1977.
[61] E. Figielska, "A heuristic for scheduling in a two-stage hybrid flowshop with renewable resources shared among the stages," *European Journal of Operational Research*, vol. 236, no. 2, pp. 433–444, 2014.
[62] B. Naderi, S. Gohari, and M. Yazdani, "Hybrid flexible flowshop problems: models and solution methods," *Applied Mathematical Modelling*, vol. 38, no. 24, pp. 5767–5780, 2014.
[63] J. Brimberg, P. Hansen, N. Mlandinovic, and E. D. Taillard, "Improvement and comparison of heuristics for solving the uncapacitated multisource weber problem," *Operations Research*, vol. 48, no. 3, pp. 444–460, 2000.
[64] J. Zhu, X. Li, and Q. Wang, "Complete local search with limited memory algorithm for no-wait job shops to minimize makespan," *European Journal of Operational Research*, vol. 198, no. 2, pp. 378–386, 2009.
[65] E. Taillard, "Benchmarks for basic scheduling problems," *European Journal of Operational Research*, vol. 64, no. 2, pp. 278–285, 1993.



Jie Zhu received her B.Sc. degree in Computer Science & Technology from Nanjing University of Post & Telecommunication, Nanjing, in 2005. Then she entered the MS and Ph.D integration program and received Ph.D. degree in Applied Computer Science from School of Computer Science and Engineering, Southeast University, Nanjing, China, in 2011. From November 2008 to November 2009, she was with Department of Electrical and Computer Engineering, University of Western Ontario, London, Ontario, Canada and Centre for Computer-assisted Construction Technologies National Research Council, London, Ontario, Canada, as a Visiting Student. She joined Nanjing University of Post & Telecommunication, Nanjing, China, in 2014, and is currently a lecturer at the School of Computer. She is the author or co-author over more than 20 academic papers, some of which have been published in international journals such as *IEEE Transactions on Automation Science and Engineering*, *European Journal of Operational Research*, *International Journal of Production Research*. Her research interests include Machine Scheduling, Project Scheduling, Workflow Optimization and Cloud Computing, among which Task Scheduling and Resource Provisioning in Clouds are her current core research areas.

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

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

Xiaolong Xu received his B.Sc. in Computer Science & Technology, M.Sc. in computer software and theories and Ph.D. degree in communications and information systems at Nanjing University of Posts & Telecommunications, Nanjing, China, in 1999, 2002 and 2008, respectively. From 2011 to 2013, he did postdoctoral research at the the School of Computer, Nanjing University of Posts & Telecommunications. He is currently a professor at the School of Computer, Nanjing University of Posts & Telecommunications. He is a senior member of China Computer Federation. His current research interests include cloud computing, mobile computing, intelligent agent and information security.