# Scheduling Periodical multi-stage jobs with fuzziness to elastic cloud resources

Jie Zhu, Xiaoping Li, *Senior Member, IEEE,* Rubén Ruiz, Wei Li, *Senior Member, IEEE,* Haiping Huang, and Albert Y. Zomaya, *Fellow, IEEE*

**Abstract**—We investigate a workflow scheduling problem with stochastic task arrival times and fuzzy task processing times and due dates. The problem is common in many real-time and workflow-based applications, where tasks with fixed stage number and linearly dependency are executed on scalable cloud resources with multiple price options. The challenges lie in proposing effective, stable and robust algorithms under stochastic and fuzzy tasks. A triangle fuzzy number based model is formulated. Two metrics are explored: the cost and the degree of satisfaction. An iterated heuristic framework is proposed to periodically schedule tasks, which consists of a task collection and a fuzzy task scheduling phases. Two task collection strategies are presented and two task prioritization strategies are employed. In order to achieve a high satisfaction degree, deadline constraints are defined at both job and task levels. By designing delicate experiments and applying sophisticated statistical techniques, experimental results show that the proposed algorithm is more effective and robust than the two existing methods.

**Index Terms**—Job scheduling, Fuzzy processing times, Fuzzy deadlines, Cloud computing.

## 1 Introduction

Multi-stage jobs, a kind of workflow application, are common in real-time and cloud environments [1], [2]. For example, there are massive speech recognition requests in many modern health-care centers. For each speech recognition job, the fixed number of linearly dependent compute-intensive stages (or tasks) are sequentially executed: pre-emphasis, windowing, Fourier transform, Mel-filter bank, IDP (interlaced derivative pattern) and classification [3]. Intermediate data is transmitted between these consecutive stages. All requests arrive stochastically. Processing times and sizes of immediate data for each request cannot be precisely given or are uncertain, i.e., they are not fully characterized before the request is executed. Since it is difficult to fulfill the massive number of compute-intensive requests by private data centers, all stages are allocated to geographically distributed cloud virtual clusters. The same stages are processed on the same cluster which initially includes only reserved VMs (which are obtained with a significant discount for long term usage, paid by months or years). Stochastically arriving requests lead to virtual clusters being frequently overloaded for which there are two possible solutions: rejecting requests [4] or renting more resources. In practice, users expect immediate response to their requests, i.e., users do not expect to be rejected. It is necessary to temporarily rent a few more expensive on-demand VMs (paid by minutes or hours). There are many other similar scenarios, e.g., NLP (Natural Language Processing), image processing, MapReduce for big data, etc.

In this paper, we consider a multi-stage job scheduling problem to both minimize the total rental cost of VMs and to maximize users' satisfaction. Jobs arrive at the system stochastically. Stages of each job are compute-intensive and linearly dependent. Each job is expected to be finished before a due date which is determined by a SLA (Service Level Agreement) [5]. Stage processing time, transmission time of immediate data and due dates are fuzzy. Stages are allocated to geographically distributed cloud virtual clusters and stages are processed by VMs of the same virtual cluster. Both reserved VMs and on-demand VMs can be provisioned to every virtual cluster. On-demand VMs are rented only when no VM is available in the virtual cluster. The pricing structures of the on-demand and the reserved VMs are different. The former ones are paid by a short time unit (by minute or hour) and the latter ones are paid by a long time term (by month or year).

The problem with fuzzy processing time, transmission time and due dates is more practical than the problem in a previous study [6] where all of the three temporal parameters are crisp. The problem with crisp parameters is already NP-hard, therefore, it follows that the problem under study is also NP-hard. Fuzziness results in great challenges for scheduling massive linearly constrained stages to geographically distributed VMs: (i) "Butterfly effects"


- *Jie Zhu is with the School of Computer Science & Technology, Nanjing University of Posts & Telecommunications, China, 210023; and also at the 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 is with 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*
- *R. Ruiz is with Grupo de Sistemas de Optimización Aplicada, Universitat Politècnica de València, Camino de Vera s/n, 46022, València, Spain (Tel: 34-96-3877007 ext.74946; Fax: 34-96-3877499). E-mail: rruiz@eio.upv.es.*
- *Haiping Huang is with the School of Computer Science & Technology, Nanjing University of Posts & Telecommunications,China,210048. E-mail: hhp@njupt.edu.cn*
- *Wei Li and Albert Y. Zomaya are with Centre of Distributed and High Performance Computing, School of Computer Science, The University of Sydney, Australia, 2006. E-mail: {weiwilson.li, albert.zomaya}@sydney.edu.au*


*Manuscript received June 15, 2020.*

might result by the fuzziness for schedules, i.e., a slightly later stage in a schedule generates even more uncertainty. Fuzziness of stages also results in fuzzy available slots on rented VMs. It is desirable to construct appropriate schedules by transferring all involved fuzziness to crisp values as early as possible. However, earlier transfer implies more computation while later transfer means more uncertainty, i.e., it is important to determine the width of transfer time windows. (ii) Stochastically arriving tasks might require more resources than the available ones. Though the total rental cost might be decreased by flexible due dates and some tolerable delays, users' satisfaction might suffer. On the contrary, users' satisfaction might be increased by renting more on-demand VMs with an increase in the total rental cost. It is difficult to strike a balance between these two conflicting objectives. (iii) Calculating fuzzy numbers requires longer processing times when compared to crisp numbers to evaluate schedules. In addition, geographically distributed VMs and linearly constrained stages make the scheduling process complex. It is crucial to develop faster algorithms for massive stochastic requests.

The paper aims to provide an effective and efficient method for the dynamic fuzzy task scheduling problem. The main contributions of this paper are summarized as follows.

(i) Considering workflows with the characteristics of stochastic and fuzzy tasks, a new fuzzy scheduling problem model is formulated for scheduling tasks to elastic cloud resources with multiple pricing structures.
(ii) A dynamic and fuzzy scheduling framework is presented which collects newly arriving tasks, generates fuzzy schedules of collected tasks, monitors task execution states and periodically updates VMs availability. A re-schedule strategy is applied to improve the schedules.
(iii) A fuzzy min-heap is employed to prioritize ready tasks, and two-dimensional orthogonal lists are applied to maintain the fuzzy idle time slots of VMs, where the feasible fuzzy time slots can be quickly found and assigned to tasks.

The rest of the paper is organized as follows. Section 2 gives the review of the related works. The considered problem is described and modeled in Section 3. Section 4 presents the proposed heuristic framework and its major components. Thorough experiments are performed to evaluate the proposal and to compare algorithms in Section 5, followed by conclusions and future research in Section 6.

## 2 Related Work

Several frameworks have been proposed for running workflows in clouds [7], [8]. Cloud workflow scheduling problems have been widely studied in the literature. The commonly considered aspects of tasks in workflows are: static or dynamic tasks, independent or precedence constrained, deterministic or non-deterministic, among others.

Some classical heuristics, e.g., reinforcement learning [9], adaptive dispatching [10], max-min cloud algorithm [11] and affinity scheduling [12] have been applied for dynamic task scheduling problems. All these algorithms are simple and fast. They are able to process millions of non-dependent tasks per minute efficiently and effectively.

Some meta-heuristics have been developed for static workflow scheduling problems with resource elasticity and task dependency. Liu et al. [13] presented a PSO-based algorithm with variable neighborhood structures for scheduling data-intensive workflows with security constraints. Taheri et al. [14] proposed a PSO-based method for the bi-objective workflow scheduling problem in hybrid clouds. Considering the temporal diversity of energy prices and execution prices in clouds, Yuan et al. [15] proposed a hybrid algorithm which integrates SA with PSO for effectively dispatching arriving tasks to multiple clouds. These meta-heuristics are effective for static workflow scheduling at the expense of considerable processing times.

For dynamic workflow scheduling problems, obtaining a feasible schedule is time-consuming even in the deterministic case. Therefore, fast heuristics are desirable. Heterogeneous Earliest Finish Time (HEFT), Min-min, Max-min and Critical-Path-on-a-processor (CPOP) [16] are the most well-known heuristics for dynamic workflow scheduling problems. Other heuristics for dynamic workflow scheduling in cloud environments take into account different characteristics, such as scalability [17], data localization [18], data transmission delay [19], bandwidth limit [20], trust [21] and prices [22]. A VND-based heuristic was proposed for dynamic cloud workflows with scalability, transmission times and VM utilization in [6].

The above studies investigated scheduling problems with deterministic task processing times and data transmission times. However, scheduling problems with both dynamic and fuzziness are more realistic. Existing studies on fuzzy scheduling are mainly manufacturing industry oriented. Chanas et al. [23] employed the Lawler's algorithm and represented processing times and due dates as L-R type fuzzy numbers with power shape functions for single machine scheduling. Balin et al. [24] proposed a robust GA for minimizing fuzzy makespan for parallel machine scheduling problems with fuzzy processing times. Yeh et al. [25] adopted triangular fuzzy numbers for processing times and due dates. They proposed a simulated annealing algorithm and a genetic algorithm for the problem with learning effects. Huang et al. [26] introduced a fuzzy model for a time-dependent project scheduling problem with fuzzy activity duration times. Abdullah et al. [28] reviewed fuzzy job shop scheduling problems (JSSPs) which were classified in terms of constraints, objectives and methods. Sakawa et al. [27] proposed a genetic algorithm for the fuzzy job shop problem with fuzzy processing times and fuzzy due dates. The major differences between the considered problem and these machine scheduling problems lie in: (i) The former deals with cloud computing resources while the latter considers manufacturing resources in plants. (ii) The resources in the considered problem are scalable and flexible whereas those of fuzzy job shop problems are fixed. (ii) Tasks of the considered problem arrive stochastically while those in fuzzy job shops are assumed to arrive at the same time.

Compared to the previous study [6], the problem studied in this paper is much more general and realistic. In other words, the problem studied in [6] is a specific case of the problem under study. All fuzzy temporal parameters of the considered problem result in fuzzy schedules and fuzzy

idle slots. More in details, the start time and the finish time of each task are uncertain before it is performed or they are gradually determined along with its execution. Therefore, the scheduling framework proposed in [6] is no longer suitable for the considered problem. It is necessary to develop fuzzy approaches to deal with uncertainty. To the best of our knowledge, the considered problem has not been studied yet and it is challenging as well as realistic.

## 3 Problem Description

In the considered problem, the fuzzy temporal parameters are represented by triangular fuzzy numbers. A triangular fuzzy number (TFN) $\widetilde{x}$ is defined as $\widetilde{x}$ =($x^{\min}$, $x^{\text{most}}$, $x^{\max}$) with $x^{\min} \leq x^{\text{most}} \leq x^{\max}$. $x^{\min}$ is the minimal value, $x^{\text{most}}$ is the most probable value and $x^{\max}$ is the maximal value. At the extreme case, a real number $x$ is represented by TFN $\widetilde{x} = (x, x, x)$. The operations to be used in the following are defined as:

$$\widetilde{x} + \widetilde{y} = (x^{\min} + y^{\min}, x^{\text{most}} + y^{\text{most}}, x^{\max} + y^{\max}) \quad (1)$$

$$\widetilde{x} - \widetilde{y} = (x^{\min} - y^{\max}, x^{\text{most}} - y^{\text{most}}, x^{\max} - y^{\min}) \quad (2)$$

$$\widetilde{x} \times \widetilde{y} = (x^{\min} \times y^{\min}, x^{\text{most}} \times y^{\text{most}}, x^{\max} \times y^{\max}) \quad (3)$$

$$\max\{\widetilde{x}, \widetilde{y}\} = (\max\{x^{\min}, y^{\min}\}, \max\{x^{\text{most}}, y^{\text{most}}\}, \max\{x^{\max}, y^{\max}\}) \quad (4)$$

$$C_1(\widetilde{x}) = (x^{\min} + 2 \times x^{\text{most}} + x^{\max})/4 \quad (5)$$

$$C_2(\widetilde{x}) = x^{\text{most}} \quad (6)$$

$$C_3(\widetilde{x}) = x^{\max} - x^{\min} \quad (7)$$

$$\Big(\forall i \in \{1,2,3\}, C_i(\widetilde{x}) = C_i(\widetilde{y})\Big) \rightarrow \widetilde{x} = \widetilde{y} \quad (8)$$

$$\Big(\exists i \in \{1,2,3\}, C_i(\widetilde{x}) < C_i(\widetilde{y}) \wedge \Big(\forall j < i, j \in \{1,2,3\}, C_j(\widetilde{x}) = C_j(\widetilde{y})\Big)\Big) \rightarrow \widetilde{x} < \widetilde{y} \quad (9)$$

$$\Big(\exists i \in \{1,2,3\}, C_i(\widetilde{x}) > C_i(\widetilde{y}) \wedge \Big(\forall j < i, j \in \{1,2,3\}, C_j(\widetilde{x}) = C_j(\widetilde{y})\Big)\Big) \rightarrow \widetilde{x} > \widetilde{y} \quad (10)$$

Notations to be used in this paper are given in Table 1.

Jobs arrive at the system stochastically which are scheduled time-window by time-window. Each time window is a scheduling period $\varepsilon$. In terms of [29], jobs arriving in a scheduling period are regarded as a Job-related Real-time Event (JRE). Suppose the system starts at time zero and $Q$ time windows are considered. The JRE arrives during the time interval $[(q-1) \times \varepsilon, q \times \varepsilon]$ and is denoted as $E_q = \{J_j | j = N_{q-1} + 1, \ldots, N_{q-1} + n_q\}$ where $n_q$ is the number of jobs in $E_q$ ($1 \leq q \leq Q$). $N_q$ is the total number of jobs that arrive at system before $q \times \varepsilon$, i.e., $N_q = N_{q-1} + n_q$.

Different from the traditional nonlinear constrained workflows which are represented by DAGs (Directed Acyclic Graph), we consider a special workflow problem in which tasks are linearly constrained in this paper. Job $J_j$ is composed of $m$ stages (or tasks) $\mathcal{T}_{j,1}$, …, $\mathcal{T}_{j,m}$ which are processed sequentially or linearly, i.e., each task $\mathcal{T}_{j,i}$ has only one immediate predecessor task $\mathcal{T}_{j,i-1}$ and only one immediate successor task $\mathcal{T}_{j,i+1}$ except the first and the last tasks of job $J_j$. The task dependency indicates that a task can start only after its immediate predecessor task is completed and the data is transmitted from its immediate predecessor task. Task interruption or preemption is not allowed which means that a task is processed continuously once it starts until its completion. The processing time of $\mathcal{T}_{j,i}$ is represented by $\widetilde{p}_{j,i} = (p_{j,i}^{\min}, p_{j,i}^{\text{most}}, p_{j,i}^{\max}), i = 1, \ldots, m$. For tasks of the same stage, VMs are provisioned with the same configuration, i.e., processing times are machine-independent. The real processing time $p_{j,i}$ of $\mathcal{T}_{j,i}$ is certain only after $\mathcal{T}_{j,i}$ is completed satisfying $p_{j,i}^{\min} \leq p_{j,i} \leq p_{j,i}^{\max}$. Setup times of tasks are negligible or included in the processing times. The transmission time of the intermediate results between $\mathcal{T}_{j,i}$ to $\mathcal{T}_{j,i+1}$ is not negligible and is denoted by $\widetilde{d}_{j,i} = (d_{j,i}^{\min}, d_{j,i}^{\text{most}}, d_{j,i}^{\max})$. The real transmission time $d_{j,i}$ is certain only after the data transmission is completed and satisfies $d_{j,i}^{\min} \leq d_{j,i} \leq d_{j,i}^{\max}$. The fuzzy processing time and the fuzzy transmission time lead to the start time $\widetilde{b}_{j,i}$ and the completion time $\widetilde{c}_{j,i}$ of $\mathcal{T}_{j,i}$ being also fuzzy numbers. The real start and completion times of $\mathcal{T}_{j,i}$ are denoted as $b_{j,i}$ and $c_{j,i}$ respectively. $b_{j,i}$ is certain once $\mathcal{T}_{j,i}$ begins to execute and $c_{j,i}$ is certain only after $\mathcal{T}_{j,i}$ is completed.

How to set the triangle values (the minimum, maximum and most probable values) for each fuzzy variable is crucial to the performance of a scheduling algorithm. These values can be obtained from historical data using statistical tools.

The same stage of all jobs is processed by the same virtu-

TABLE 1
Notations to be used.

| Notation | Description |
|---|---|
| $\varepsilon$ | Scheduling period width |
| $t$ | Current time |
| $Q$ | Number of scheduling periods/events |
| $m$ | Number of stages in a job |
| $E_q$ | $q^{th}$ JRE in the $q^{th}$ scheduling period |
| $J_j$ | $j^{th}$ job |
| $t_j$ | Arriving time of $J_j$ |
| $n_q$ | Number of jobs in $E_q$ |
| $\mathcal{T}_{j,i}$ | $i^{th}$ task of $J_j$, $i = 1, \ldots, m$ |
| $\widetilde{p}_{j,i}$ | Fuzzy processing time of $\mathcal{T}_{j,i}$ |
| $p_{j,i}^{\min}$, $p_{j,i}^{\text{most}}$, $p_{j,i}^{\max}$ | Minimal/most probable/maximal value of $\widetilde{p}_{j,i}$ |
| $p_{j,i}$ | Real processing time of $\mathcal{T}_{j,i}$ |
| $\widetilde{d}_{j,i}$ | Fuzzy transmission time from $\mathcal{T}_{j,i}$ to $\mathcal{T}_{j,i+1}$ |
| $d_{j,i}^{\min}$,$d_{j,i}^{\text{most}}$, $d_{j,i}^{\max}$ | Minimal/most probable/maximal value of $\widetilde{d}_{j,i}$ |
| $d_{j,i}$ | Real transmission time from $\mathcal{T}_{j,i}$ to $\mathcal{T}_{j,i+1}$ |
| $\widetilde{b}_{j,i}$ | Fuzzy start time of $\mathcal{T}_{j,i}$ |
| $b_{j,i}^{\min}$, $b_{j,i}^{\text{most}}$, $b_{j,i}^{\max}$ | Minimal/most probable/maximal value of $\widetilde{b}_{j,i}$ |
| $b_{j,i}$ | Real start time of $\mathcal{T}_{j,i}$ |
| $\widetilde{c}_{j,i}$ | Fuzzy completion time of $\mathcal{T}_{j,i}$ |
| $c_{j,i}^{\min}$, $c_{j,i}^{\text{most}}$, $c_{j,i}^{\max}$ | Minimal/most probable/maximal value of $\widetilde{c}_{j,i}$ |
| $c_{j,i}$ | Real completion time of $\mathcal{T}_{j,i}$ |
| $\widetilde{D}_j = (D_j^1, D_j^2)$ | Fuzzy due date of $J_j$ |
| $VC_i$ | Virtual cluster for processing the $i^{th}$ stage of jobs |
| $\psi_i^r$ | Unit price for reserved VMs in $VC_i$ |
| $\psi_i^o$ | Unit price for on-demand VMs in $VC_i$ |
| $u$ | Time unit for charging on-demand VMs |
| $\mathbb{R}_i$ | Reserved VM set in $VC_i$ |
| $n_i^r$ | Number of reserved VMs in $VC_i$ |
| $\mathbb{O}_i$ | On-demand VM set in $VC_i$ |
| $n_i^o$ | Number of on-demand VMs in $VC_i$ |
| $s_{j,i}(t)$ | Assignments of $\mathcal{T}_{j,i}$ at time $t$ |
| $v_{j,i}$ | VM assigned to $\mathcal{T}_{j,i}$ |
| $\mathcal{F}(t)$ | Fuzzy Schedule at time $t$ |
| $N(t)$ | Number of arrived jobs at time $t$ |
| $\mathcal{F}^{\text{fuzzy}}(t)$, $\mathcal{F}^{\text{semi}}(t)$,$\mathcal{F}^{\text{real}}(t)$ | Set of fuzzy/semi-fuzzy/real assignments at time $t$ |

al cluster which contains a set of VMs. Since virtual clusters are geographically distributed, all stages are distributed to VMs. Every stage is allocated to a specific VM and all the stages on the same VM are queued.

A VM is a light-weight implementation of the execution environment. VMs in a virtual cluster are homogeneous. Each VM can process only one task at a time, i.e., multi-tenant or multi-thread scenarios are not considered. Since VMs are light-weight, the provisioning and launching time of a VM is considered to be negligible. $VC_i$ denotes the virtual cluster where the $i^{th}$ stage ($i = 1, 2, \ldots, m$) of each job is processed. Each virtual cluster includes two types of VMs: reserved VM and on-demand VM. The sets of the reserved and on-demand VMs in $VC_i$ are denoted as $\mathbb{R}_i$ and $\mathbb{O}_i$ respectively. $n_i^r$ and $n_i^o$ are the numbers of VMs in $\mathbb{R}_i$ and $\mathbb{O}_i$, i.e., $n_i^r = |\mathbb{R}_i|$ and $n_i^o = |\mathbb{O}_i|$. Reserved VMs are available for a long period of time which is regarded as a constant. $n_i^o$ is initialized as 0 and it changes along with the workloads. An on-demand VM is rented and paid by a short time unit $u$ (e.g., by hour or a quarter of an hour). On-demand VMs might be needed at different time points, i.e., there would be some idle time intervals for renting on-demand VMs. It is necessary to make full use of every rented $u$ and try to release time units in idle time intervals to minimize the total renting cost. Suppose the unit prices of on-demand and reserved VMs in $VC_i$ are $\psi_i^o$ and $\psi_i^r$ respectively. Generally, $\psi_i^r$ is much cheaper than $\psi_i^o$ (e.g., the reserved pricing is up to 75% cheaper at Amazon compared to the on-demand pricing).

The assignment of $\mathcal{T}_{j,i}$ changes over time in three states: fuzzy, semi-fuzzy and real. Figure 1 shows a partial Gantt chart of a fuzzy schedule at time $t$. (i) Before the actual task execution, the start and completion times of $\mathcal{T}_{j,i}$ are fuzzy due to the fuzzy processing and transmission times. The assignment with the fuzzy start and completion times is called the fuzzy assignment. The fuzzy assignment of $\mathcal{T}_{j,i}$ at current time $t$ is represented by a 3-tuple $s_{j,i}(t) =< \widetilde{b}_{j,i}, \widetilde{c}_{j,i}, v_{j,i} >$ where $v_{j,i} \in VC_i$, $b_{j,i}^{\min} > t$ and $\widetilde{c}_{j,i} = \widetilde{b}_{j,i} + \widetilde{p}_{j,i}$ as the gray blocks show in Figure 1. The variable $v_{j,i}$ represents the VM assigned to $\mathcal{T}_{j,i}$. (ii) When $\mathcal{T}_{j,i}$ is being executed, its start time is certain and the completion time is fuzzy. The assignment with the real start and fuzzy completion times is called the semi-fuzzy assignment. The semi-fuzzy assignment of $\mathcal{T}_{j,i}$ is $s_{j,i}(t) =< b_{j,i}, \widetilde{c}_{j,i}, v_{j,i} >$, where $b_{j,i} \leq t < c_{j,i}^{\max}$ and $\widetilde{c}_{j,i} = b_{j,i} + \widetilde{p}_{j,i}$ as illustrated as the shaded blocks in Figure 1. (iii) When $\mathcal{T}_{j,i}$ is completed, its processing and completion times are certain, the real assignment of $\mathcal{T}_{j,i}$ is $s_{j,i}(t) =< b_{j,i}, c_{j,i}, v_{j,i} >$ where $t \geq c_{j,i}$ and $c_{j,i} = b_{j,i} + p_{j,i}$ as demonstrated as the white blocks in Figure 1.

The assignment of $\mathcal{T}_{j,i}$ at time $t$ is fuzzy if its immediate predecessor task on $v_{j,i}$ is not completed or the data transmission from its predecessor task is not completed. Suppose $\mathcal{T}_{j',i}$ is the immediate predecessor task of $\mathcal{T}_{j,i}$ on $v_{j,i}$. $\widetilde{b}_{j,i}$ is calculated by

$$\widetilde{b}_{j,i} = \max\{\widetilde{c}_{j,i-1} + \widetilde{d}_{j,i-1}, \widetilde{c}_{j',i}\}. \tag{11}$$

The assignment of $\mathcal{T}_{j,i}$ at the current time $t$ is semi-fuzzy if the data transmission from its predecessor task $\mathcal{T}_{j,i-1}$ is completed and its immediate predecessor task $\mathcal{T}_{j',i}$ on $v_{j,i}$ is completed. The state of $s_{j,i}(t)$ is changed from fuzzy to semi-fuzzy immediately at the time when the real start time can be obtained. Once $\mathcal{T}_{j,i}$ is completed at time $t'$, $s_{j,i}(t)$ ($t \geq t'$) is real.

The schedule $\mathcal{F}(t)$ is the assignment of the set of the arrived tasks at time $t$ to VMs, i.e., $\mathcal{F}(t) = \{s_{j,i}(t) | 1 \leq j \leq N(t), i = 1, \ldots, m\}$ where $N(t)$ is the total number of arrived jobs at time $t$. $\mathcal{F}^{\text{fuzzy}}(t)$, $\mathcal{F}^{\text{semi}}(t)$ and $\mathcal{F}^{\text{real}}(t)$ are the subsets of fuzzy, semi-fuzzy, and real task assignments at time $t$ respectively.

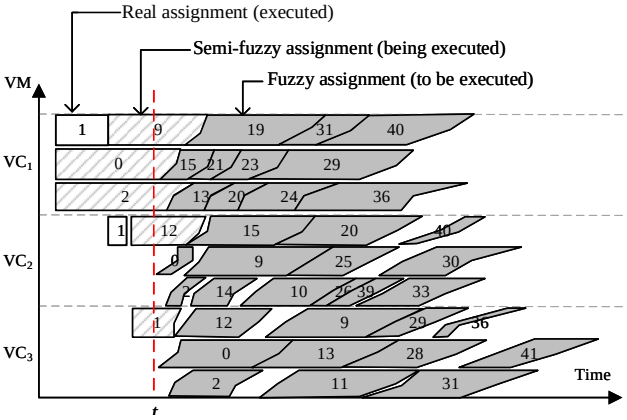


Fig. 1. Gantt chart for a fuzzy schedule at time $t$

The schedule $\mathcal{F}(t)$ is predictive which is actually executed according to the task order and the VM assignments. As time goes, fuzzy assignments are continuously updated to semi-fuzzy assignments (being executed) or some even become real assignments (completed).

The objective is to maximize users' satisfaction and to minimize the total rental cost. Similar to [30], the due date of $J_j$ is described by a duple $\widetilde{D}_j = (D_j^1, D_j^2)$ ($D_j^1 < D_j^2$) which represents the satisfaction degree with respect to the job completion time $c_{j,m}$. The real satisfaction degree $SD_j$ of $J_j$ is defined by Equ. (12). Similarly, given a fuzzy job completion time $\widetilde{c}_{j,m}$, a fuzzy satisfaction degree $\widetilde{SD}_j$ of $J_j$ is defined by Equ. (13).

$$SD_j = \begin{cases} 1, & c_{j,m} \leq D_j^1 \\ 1 - \frac{c_{j,m} - D_j^1}{D_j^2 - D_j^1}, & D_j^1 < c_{j,m} < D_j^2 \\ 0, & c_{j,m} \geq D_j^2. \end{cases} \tag{12}$$

$$\widetilde{SD}_j = \min\left\{1, \max\left\{0, 1 - \frac{\widetilde{c}_{j,m} - D_j^1}{D_j^2 - D_j^1}\right\}\right\} \tag{13}$$

Since the considered problem is bi-objective, we apply a LWS (Linear Weighted Sum) method to evaluate solutions. Equ. (14) is the LWS of the total rental cost $C(\widetilde{T})$ and the average satisfaction degree $S(\widetilde{T})$.

$$LWS = w \times \frac{\overline{C}(\widetilde{T})}{C(\widetilde{T})} + (1 - w) \times S(\widetilde{T}) \tag{14}$$

$w$ ($w \in (0, 1)$) is the weight coefficient to control the tradeoff between cost and the satisfaction. A larger $LWS$ ($LWS \in (0, 1]$) implies a better solution. $\widetilde{T}$ is the total scheduling time. $\overline{C}(\widetilde{T})$ is the lower bound of $C(\widetilde{T})$ for

normalizing $C(\widetilde{T})$. $S(\widetilde{T})$ and $C(\widetilde{T})$ is defined by Equ.(15) and Equ.(16) respectively.

$$S(\widetilde{T}) = \frac{1}{N(\widetilde{T})} \sum_{j=1}^{N(\widetilde{T})} \min\left\{1, \max\left\{0, 1 - \frac{\widetilde{c}_{j,m} - D_j^1}{D_j^2 - D_j^1}\right\}\right\} \tag{15}$$

$$C(\widetilde{T}) = C^r(\widetilde{T}) + C^o(\widetilde{T}) \tag{16}$$

$$C^r(\widetilde{T}) = \sum_{i=1}^{m} n_i^r \times \left\lceil \frac{\widetilde{T}}{u} \right\rceil \times \psi_i^r \tag{17}$$

$$C^o(\widetilde{T}) = \sum_{i=1}^{m} \sum_{v \in \mathbb{O}_i} \left\lceil \frac{\max\limits_{j=1}^{N(\widetilde{T})}\{\widetilde{c}_{j,i} | v_{j,i} = v\} - \min\limits_{j=1}^{N(\widetilde{T})}\{\widetilde{b}_{j,i} | v_{j,i} = v\}}{u} \right\rceil \times \psi_i^o \tag{18}$$

$$\widetilde{b}_{j,i} - \max_{j=1}^{N(\widetilde{T})} \left\{\widetilde{c}_{j',i} | v_{j,i} = v_{j',i} \in \mathbb{O}_i, \widetilde{c}_{j',i} < \widetilde{b}_{j,i}\right\} < u \tag{19}$$
$$\forall j, j' = 1, \ldots N(\widetilde{T}), i = 1, \ldots, m$$

$C(\widetilde{T})$ is composed of the cost on the reserved VMs $C^r(\widetilde{T})$ and the cost on the on-demand VMs $C^o(\widetilde{T})$. $C^r(\widetilde{T})$ and $C^o(\widetilde{T})$ are defined by Equ.(17) and Equ.(18), respectively. Equ.(18) indicates that on-demand VMs are charged by time units, i.e., a period less than $u$ is still charged by $u$. Constraint (19) implies an on-demand VM with an idle interval between two successive tasks not less than $u$ might be released. A higher $S(\widetilde{T})$ indicates a better user experience. A lower $C(\widetilde{T})$ means higher utilization on reserved VMs and a less need for on-demand VMs.

$\mathcal{F}(\widetilde{T})$ is feasible if and only if the constraints (20) and (21) are satisfied.

$$\max\{\widetilde{b}_{j,i}, \widetilde{b}_{j',i}\} \geq \min\{\widetilde{c}_{j,i}, \widetilde{c}_{j',i}\} \tag{20}$$

$$\widetilde{b}_{j,i} \geq \max\{t_j, \widetilde{b}_{j,i-1} + \widetilde{p}_{j,i-1} + \widetilde{d}_{j,i-1}\} \tag{21}$$

Constraint (20) guarantees no overlapping among tasks on the same VM. Constraint (21) defines the precedence constraint, i.e., a task cannot start until it receives the intermediate result from its immediate predecessor. It also indicates that a job cannot start before its arrival time.

## 4 Proposed Algorithms

The considered dynamic scheduling procedure is completely reactive as a result of three aspects: (1) arrival time of jobs is uncertain; (2) transmission times between any two consecutive tasks are uncertain until the data transmission is completed; (3) task processing times are uncertain until tasks are completed. Fuzzy tasks are scheduled event by event at time $t = \varepsilon, 2\varepsilon, \ldots, Q \times \varepsilon$. The Fuzzy Dynamic Event Scheduling (FDES) framework for scheduling dynamic fuzzy events is detailed in Algorithm 1 (as depicted in Fig. 2).

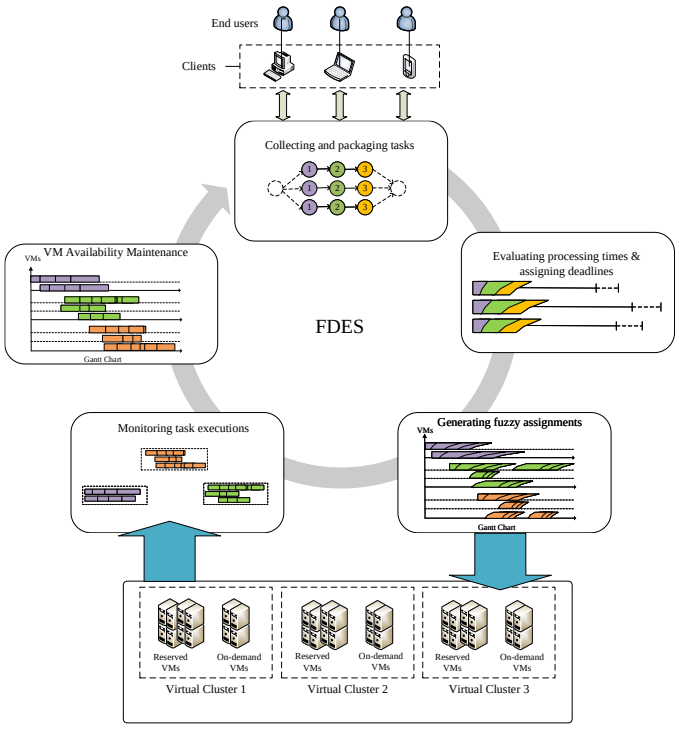


Fig. 2. Fuzzy Dynamic Event Scheduling (FDES) framework

A feasible fuzzy schedule $\mathcal{F}(t)$ is initially empty at time $t = 0$ and updated iteratively at time $t = \varepsilon, 2\varepsilon, \ldots, Q \times \varepsilon$ (Lines 3-6):

1) The execution progress of task is monitored and the availability of VMs is updated (Line 3). After some tasks are completed, their assignments are updated from semi-fuzzy to real. After some tasks are started, their assignments are updated from fuzzy to semi-fuzzy.
2) Tasks to be scheduled are collected at time $t$ by TCS (Task Collection Strategy) (Line 4).
3) Deadlines of collected tasks are computed by TDA (Task Deadline Assignment) (Line 5).
4) Fuzzy assignments for collected tasks are generated by FTS (Fuzzy Task Scheduling)(Line 6).

FDES consists in four main components: the VM Availability Maintenance (VMAM), the Task Collection Strategy (TCS), the Task Deadline Assignment (TDA), and the Fuzzy Task Scheduling (FTS). Details are introduced in the following sections.

### 4.1 VM Availability Maintenance (VMAM)

A time slot occupied by $J_j$ on VM $v$ is represented by $< v, J_j, [\widetilde{\mathfrak{a}}, \widetilde{\mathfrak{b}}] >$, where $[\widetilde{\mathfrak{a}}, \widetilde{\mathfrak{b}}]$ is a fuzzy time interval. $< v, 0, [\widetilde{\mathfrak{a}}, \widetilde{\mathfrak{b}}] >$ indicates an idle time slot on $v$. The initially fuzzy time slot of $v$ is $< v, 0, [(0,0,0), (\infty,\infty,\infty)] >$. After some tasks are scheduled, some idle time slots might be separated on each VM.

A VM availability maintenance method is proposed to store time slots in each virtual cluster. We use a two-dimensional orthogonal list to preserve idle and occupied time slots of each virtual cluster. Each node of the orthogonal list represents a time slot and is right-linked to the

**Algorithm 1:** Proposed Fuzzy Dynamic Event Scheduling (FDES) framework

```
for q = 1 to Q do
    t = q × ε;
    Monitoring the execution of tasks and updating the availability of VMs;
    Collecting tasks to be scheduled in the current time window [t − ε, t] and packaging them into E_q;
    Computing deadlines of tasks in E_q;
    Generating fuzzy assignments for tasks in E_q;
Computing S(t) and C(t);
return S(t), C(t).
```

adjacent time slot (either idle or occupied) on the same VM. For the $k^{th}$ VM in $VC_i$, the head of the right-linked nodes is preserved in $\mathbb{L}_{i,k}$. The nodes representing idle time slots in the same virtual cluster are also down-linked in ascending order of their start times. Idle fuzzy time slots for reserved VMs and on-demand VMs are down-linked separately. In each virtual cluster $VC_i$ $(i = 1, \ldots, m)$, $\mathbb{I}_i^{\mathrm{r}}$ maintains the head idle time slot of the reserved VMs and $\mathbb{I}_i^{\mathrm{o}}$ maintains that of the on-demand ones. Fig. 3 depicts an example of the orthogonal list for maintaining the fuzzy time slots. For simplicity, only the time intervals are shown in the nodes of the orthogonal list.

The length of a fuzzy time slot $< v, J_j, [\widetilde{\mathfrak{a}}, \widetilde{\mathfrak{b}}] >$ is $\widetilde{\mathfrak{b}} - \widetilde{\mathfrak{a}}$. The nodes with a length of less than or equal to zero should be removed from the lists (it is possible to have a time interval $[\widetilde{\mathfrak{a}}, \widetilde{\mathfrak{b}}]$ with $\widetilde{\mathfrak{a}} < \widetilde{\mathfrak{b}}$ but $\widetilde{\mathfrak{b}} - \widetilde{\mathfrak{a}} < 0$ due to the fuzzy operations, e.g., $[(3, 4, 5), (2, 4, 6)]$). The orthogonal lists are updated in real time according to the actual execution of tasks. Any node with a time interval $[\widetilde{\mathfrak{a}}, \widetilde{\mathfrak{b}}]$ less than the current time $t$ ($\widetilde{\mathfrak{b}} \leq t$) is removed from the corresponding list since these time slots indicate the past assignments with real states. Only the fuzzy and semi-fuzzy time slots are needed.

### 4.2 Task Collection Strategy (TCS)

A general and simple strategy for the task collection consists of collecting new tasks arriving in the past time window. This may result in a low-quality solution. Some tasks may be completed earlier or later than expected which results in the fuzzy schedule produced in the last time interval to be invalid.

We propose two TCS candidates to determine tasks to be scheduled: (1) TCS1 collects only new tasks arriving during the last scheduling period. (2) TCS2 undoes the fuzzy assignments of unstarted tasks and combines them with the newly arrived tasks during the previous time window. The fuzzy assignments of the undone tasks are removed from the current $\mathcal{F}^{\text{fuzzy}}(t)$ and the fuzzy time intervals occupied by these tasks are returned to the corresponding orthogonal lists.

Table 2 shows an example of the arrival of new jobs in five scheduling periods. It also lists the jobs collected by TCS1 and TCS2 at each period. Fig. 4 and 5 show the Gantt charts of the fuzzy schedule $\mathcal{F}(t)$ $(t = 4\varepsilon, 5\varepsilon)$ by applying TCS1 and TCS2 respectively. The fuzzy schedule $\mathcal{F}(t)$ is generated by simply arranging the collected tasks as early as possible. We can find out that after 5 scheduling periods, the schedule applying TCS1 need four on-demand VMs in total, whereas the schedule applying TCS2 needs only two. In other words, collecting and rescheduling unstarted tasks in previous events might obtain better solutions.

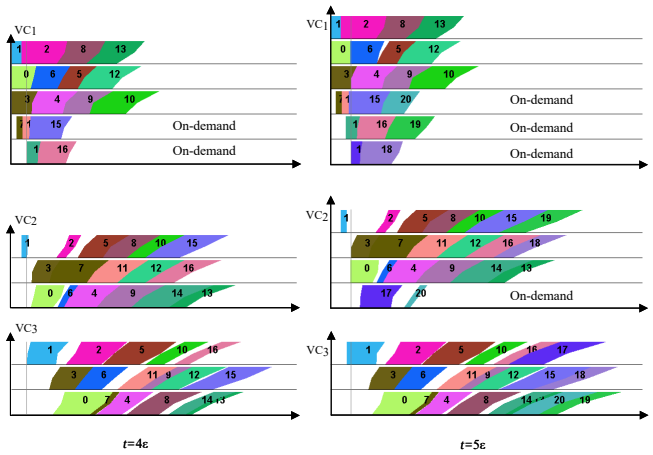


Fig. 4. Gantt charts of the fuzzy schedule $\mathcal{F}(t)$ $(t = 4\varepsilon, 5\varepsilon)$ applying TCS1.

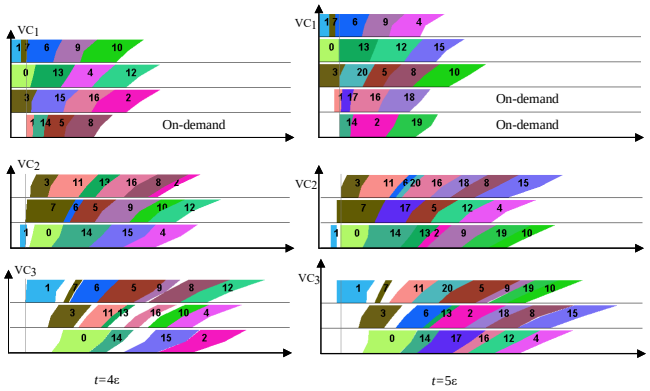


Fig. 5. Gantt charts of the fuzzy schedule $\mathcal{F}(t)$ $(t = 4\varepsilon, 5\varepsilon)$ applying TCS2.

### 4.3 Task Deadline Assignment (TDA)

Although each job has a due date, it is difficult to guarantee the hard deadline constraint because of the fuzzy temporal variables (the task processing times and the transmission times). In order to obtain solutions with a high satisfaction degree, all tasks and jobs are assigned triangular fuzzy deadlines according to Equ.(15). Given the due date $(D_j^1, D_j^2)$, a triangular fuzzy job deadline $\widetilde{\delta}_j$ of $J_j$ is constructed by the difference $\Delta\widetilde{\delta}_j$ (Equ.(22)) between the given $D_j^2$ and $D_j^1$, as defined by Equ.(23).

$$\Delta\widetilde{\delta}_j = D_j^2 - D_j^1 \tag{22}$$

$$\widetilde{\delta}_j = (D_j^1 - \Delta\widetilde{\delta}_j, D_j^1, D_j^1 + \Delta\widetilde{\delta}_j) \tag{23}$$

The task deadline (TD) is constructed based on the job deadline. Given the fuzzy job deadline $\widetilde{\delta}_j$, two kinds of task

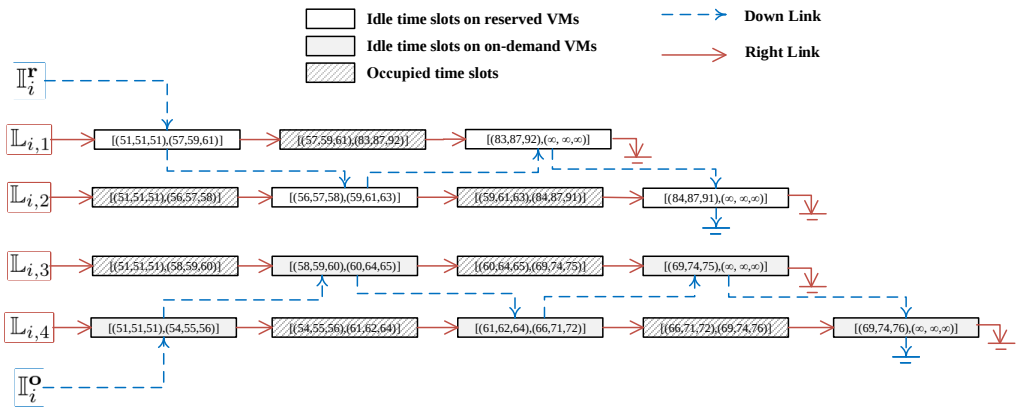


Fig. 3. An orthogonal list for storing idle and occupied time slots in a virtual cluster

TABLE 2
Examples of TCS candidates at time $t = \varepsilon, 2\varepsilon, 3\varepsilon, 4\varepsilon, 5\varepsilon$ (jobs with $*$ are old ones of which some tasks are rescheduled).

| $t$ | Newly arriving jobs | Jobs collected by TCS1 | Jobs collected by TCS2 |
|---|---|---|---|
| $\varepsilon$ | $J_0, J_1, J_2, J_3$ | $E_1 = \{J_0, J_1, J_2, J_3\}$ | $E_1 = \{J_0, J_1, J_2, J_3\}$ |
| $2\varepsilon$ | $J_4, J_5, J_6, J_7$ | $E_2 = \{J_4, J_5, J_6, J_7\}$ | $E_2 = \{J_4, J_5, J_6, J_7, J_1^*, J_2^*, J_3^*\}$ |
| $3\varepsilon$ | $J_8, J_9, J_{10}, J_{11}, J_{12}$ | $E_3 = \{J_8, J_9, J_{10}, J_{11}, J_{12}\}$ | $E_3 = \{J_8, J_9, J_{10}, J_{11}, J_{12}, J_2^*, J_4^*, J_5^*, J_6^*\}$ |
| $4\varepsilon$ | $J_{13}, J_{14}, J_{15}, J_{16}$ | $E_4 = \{J_{13}, J_{14}, J_{15}, J_{16}\}$ | $E_4 = \{J_{13}, J_{14}, J_{15}, J_{16}, J_2^*, J_4^*, J_5^*, J_6^*, J_7^*, J_8^*, J_9^*, J_{10}^*, J_{11}^*, J_{12}^*\}$ |
| $5\varepsilon$ | $J_{17}, J_{18}, J_{19}, J_{20}$ | $E_5 = \{J_{17}, J_{18}, J_{19}, J_{20}\}$ | $E_5 = \{J_{17}, J_{18}, J_{19}, J_{20}, J_2^*, J_4^*, J_5^*, J_6^*, J_8^*, J_9^*, J_{10}^*, J_{11}^*, J_{12}^*, J_{13}^*, J_{15}^*, J_{16}^*\}$ |

deadlines are introduced: RTD (the relaxed task deadline) and TTD (the tight task deadline). RTD of $\mathcal{T}_{j,i}$ is computed by Equ. (24) which assigns the deadline of a task as late as possible. TTD of $\mathcal{T}_{j,i}$ is computed by Equ. (25), where $df_j$ is the fuzzy deadline factor. TTD assigns the deadline of a task proportionally to its processing time.

$$\widetilde{\delta}_{j,i} = \widetilde{\delta}_j - \sum_{i'=i+1}^{m} \widetilde{p}_{j,i'} \tag{24}$$

$$\widetilde{\delta}_{j,i} = \begin{cases} \widetilde{\delta}_j, & i = m \\ \widetilde{\delta}_j - \lfloor df_j \times \sum_{i'=i+1}^{m} \widetilde{p}_{j,i'} \rfloor, & 1 \le i < m. \end{cases} \tag{25}$$

$$df_j = \left(\frac{\delta_j^{\min} - t_j}{\sum_{i=1}^{m} p_{j,i}^{\min}} + \frac{\delta_j^{\text{most}} - t_j}{\sum_{i=1}^{m} p_{j,i}^{\text{most}}} + \frac{\delta_j^{\max} - t_j}{\sum_{i=1}^{m} p_{j,i}^{\max}}\right)/3 \tag{26}$$

Table 3 shows an example of the task deadline assignment. Generally, TTD is less than RTD, i.e., it is "tighter". Given the job deadline and the task deadline assignment, the deadline constraint in Equ. (27) should be satisfied to obtain the solution with the high user satisfaction.

$$\widetilde{c}_{j,i} \le \widetilde{\delta}_{j,i} \tag{27}$$

TABLE 3
An example of the TDA with $t_j = 0$.

| $\widetilde{D}_j$ | $\widetilde{\delta}_j$ | $df_j$ | $i$ | $\widetilde{p}_{j,i}$ | $\widetilde{\delta}_{j,i}$ (RTD) | $\widetilde{\delta}_{j,i}$ (TTD) |
|---|---|---|---|---|---|---|
| (26,30) | (22,26,30) | 1.61 | 1 | (5,7,9) | (8,16,24) | (0,10,21) |
| | | | 2 | (4,6,8) | (16,22,28) | (13,20,27) |
| | | | 3 | (2,4,6) | (22,26,30) | (22,26,30) |

### 4.4 Fuzzy Task Scheduling (FTS)

Scheduling tasks in each event with fuzzy processing time, transmission time and due dates is a fuzzy event scheduling problem, which is solved by FTS (Fuzzy Task Scheduling). The fuzzy event scheduling problem is different from the traditional Fuzzy Job Shop Scheduling [28] and Stochastic Multi-Stage Job Scheduling [6] in three fundamental aspects: (i) The available time intervals of a VM are fuzzy and may be inconsistent with reality. (ii) Fuzzy transmission times are considered. (iii) Deadlines of tasks are also fuzzy.

In the fuzzy event scheduling problem, $n_q \times m$ tasks in $E_q$ ($q = 1, \ldots, Q$) are scheduled by FTS (as depicted in Algorithm 2) in the precedence order of the tasks. A task is ready to be scheduled only after its immediate predecessor task has been scheduled. A priority queue $\mathcal{PQ}$ is used to keep ready tasks in the correct order. Tasks in $\mathcal{PQ}$ are allocated iteratively until $\mathcal{PQ}$ is empty (Lines 2-7):

(i) Ready tasks are prioritized in $\mathcal{PQ}$ (Line 3).
(ii) The task with the highest priority (called the root task) is allocated and removed from $\mathcal{PQ}$ (Line 5).
(iii) The immediate successor task of the root task is added to the appropriate location in $\mathcal{PQ}$ (Line 7).

FTS has two major operations: a *task prioritizing* (Line 3) operation for computing the priorities of ready tasks, and a *task allocation* (Line 5) operation for generating the fuzzy assignment for a ready task.

#### 4.4.1 Task Prioritizing

The ready tasks in $\mathcal{PQ}$ are changing dynamically during the scheduling procedure. All ready tasks are scheduled sequentially, i.e., they are scheduled in some order. How to sort to tasks in $\mathcal{PQ}$ is crucial to the performance of the

**Algorithm 2:** Fuzzy Task Scheduling (FTS)

```
1 Add ready tasks of E_q to PQ;
2 while (PQ! = ∅) do
3     Prioritize and sort ready tasks in PQ;
4     Remove the root task T_{j,i} of PQ;
5     Allocate T_{j,i} with a fuzzy assignment ;
6     if i < m then
7         Append the ready task T_{j,i+1} to PQ;
8 return obtained fuzzy assignments.
```

proposed methodology. In this paper, we adopt the min-heap data structure to obtain the order tasks in $\mathcal{PQ}$. The task with the highest priority is located at the root node. If the root task is completed, it is removed from the root and its immediate successor task is appended. By adjusting the min-heap, the task with the current highest priority is placed to the root and the appended task is adjusted to the appropriate place in the min-heap tree. Therefore, how to determine the priorities of ready tasks is key to construct the min-heap. In this paper, we propose three priority strategies based on the fuzzy temporal parameters of tasks and the corresponding min-heap is called fuzzy heap.

- Task Priority 1 (TP1): the priority of a ready task is a duple consisting of two values: the possible earliest start time (PEST) of the task and the task deadline. The PEST of a ready task is determined by the fuzzy completion time of its immediate predecessor task (which has been arranged in the previous iterations) and the fuzzy transmission time, i.e., PEST($\mathcal{T}_{j,i}$)=$\widetilde{c}_{j,i-1} + \widetilde{d}_{j,i-1}$. For a Stage 1 task, its PEST is its arrival time. The task with the minimum PEST is considered to have the highest priority. If some tasks have the same PEST value, the one with the minimum task deadline has the highest priority.
- Task Priority 2 (TP2): the priority of a ready task is a duple consisting of two values: the feasible earliest start time (FEST) of the task and the task deadline. The FEST is obtained by tentatively searching on the orthogonal lists for the first idle time slot that is feasible for the task. The FEST calculating as depicted by Algorithm 3. An assignment is feasible only if both the deadline and the precedence constraints are satisfied. If no feasible assignment can be obtained, then the FEST is $+\infty$ which indicates that a new on-demand VM is needed for the task. The task with the minimum FEST has the highest priority. If many tasks have the same FEST, the one with the minimum task deadline has the highest priority. An extreme case is that the first feasible idle time slots for some ready tasks might come from the same VM and they are overlapped. Once one of the tasks is arranged to its FEST, the other conflicting tasks (with overlapped FESTs) should be re-computed, and the min-heap is adjusted correspondingly. Therefore, the operations (inserting, deleting, adjusting ) on fuzzy min-heaps for TP2 are different from those for TP1 because of the fuzziness of nodes. As shown in Fig. 6, the tasks with the overlapped FESTs are linked. Once the task with the top priority is removed, its linked tasks should be reprioritized, i.e., their FEST are re-computed. After a new ready task is inserted, all conflicting tasks should be adjusted accordingly.
- Task Priority 3 (TP3): the priority of a ready task $\mathcal{T}_{j,i}$ depends on its urgency $u_{j,i}$, which is calculated by the difference between the feasible earliest completion time of the task and the task deadline, i.e., $u_{j,i} = \delta_{j,i} - (\text{FEST}(\mathcal{T}_{j,i}) + \widetilde{p}_{j,i})$. The task with the minimum $u_{j,i}$ value has the highest priority. Since FEST is also used for prioritizing, TP3 applies the similar fuzzy min-heap as TP2 does.

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

```
1 Search right-linked idle time slots from L_i^r to find the
  first feasible idle time slot [ã, b̃] with
  max{c̃_{j,i-1} + d̃_{j,i}, ã} + p̃_{j,i} ≤ min{δ̃_{j,i}, b̃};
2 if [ã, b̃] is found then
3     return max{c̃_{j,i-1} + d̃_{j,i}, ã};
4 else
5     Search right-linked idle time slots from L_i^o to find
      the first feasible idle time slot [ã, b̃] with
      max{c̃_{j,i-1} + d̃_{j,i}, ã} + p̃_{j,i} ≤ min{δ̃_{j,i}, b̃};
6     if [ã, b̃] is found then
7         return max{c̃_{j,i-1} + d̃_{j,i}, ã};
8     else
9         return +∞;
```

Though PEST can be calculated quickly, it may be infeasible because the availability of VMs is not considered. FEST is feasible, but it needs more CPU time to obtain feasible start times.

#### 4.4.2 Task Arranging Operation

Ready tasks are arranged by FARRNGE procedure of Algorithm 4. Given a ready task $\mathcal{T}_{j,i}$, it tries to find the first feasible idle time slot $[\widetilde{a}, \widetilde{b}]$ on $\mathbb{L}_i^{\mathbf{r}}$ satisfying both the precedence and the deadline constraints, i.e., $\max\{\widetilde{c}_{j,i-1} + \widetilde{d}_{j,i}, \widetilde{a}\} + \widetilde{p}_{j,i} \leq \min\{\widetilde{\delta}_{j,i}, \widetilde{b}\}$ (Line 1). If the satisfied time slot is found, then it is assigned to the ready task. The orthogonal list of $VC_i$ is updated after the assignment (Lines 2-10). If the search fails, FARRANGE performs a similar search on the orthogonal list $\mathbb{L}_i^{\mathbf{o}}$ (Line 12). The orthogonal list of $VC_i$ is updated after the assignment (Lines 13-21). If the search on both reserved and on-demand VMs fails, a new on-demand VM is rented to which $\mathcal{T}_{j,i}$ is allocated (Lines 23-24). The orthogonal list of $VC_i$ is updated after the assignment (Lines 25-29). The feasible assignment that $\mathcal{T}_{j,i}$ is assigned to $v_{j,i}$ from the fuzzy time $\widetilde{b}_{j,i}$ to $\widetilde{c}_{j,i}$ is returned (Line 30).

#### 4.4.3 Time Complexity Analysis

The time complexity of the task prioritization operation is determined by the time complexity of computing the priority for a task and adjusting the priority queue $\mathcal{PQ}$. Suppose $\mathcal{PQ}$ is implemented by a minimum heap. The time complexity for adjusting the minimum heap is $O(\log n_q)$.

If TP1 is applied, then time complexity for computing the priority of a task is $O(1)$. In total, the minimum heap

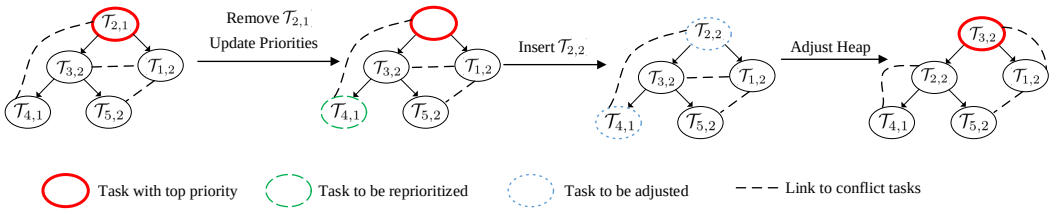


Fig. 6. An example of the fuzzy min-heap in the task prioritizing operation TP2

**Algorithm 4:** FARRANGE($\mathcal{T}_{j,i}, t$)

Search $\mathbb{L}_i^{\mathrm{r}}$ to find the first feasible idle time slot $[\widetilde{\mathfrak{a}}, \widetilde{\mathfrak{b}}]$ with $\max\{\widetilde{c}_{j,i-1} + \widetilde{d}_{j,i}, \widetilde{\mathfrak{a}}\} + \widetilde{p}_{j,i} \leq \min\{\widetilde{\delta}_{j,i}, \widetilde{\mathfrak{b}}\}$;
**if** $< v, -1, [\widetilde{\mathfrak{a}}, \widetilde{\mathfrak{b}}] >$ *is found* **then**
  $\widetilde{b}_{j,i} \leftarrow \max\{\widetilde{c}_{j,i-1} + \widetilde{d}_{j,i}, \widetilde{\mathfrak{a}}\}$; $\widetilde{c}_{j,i} \leftarrow \widetilde{b}_{j,i} + \widetilde{p}_{j,i}$;
  Remove $< v, -1, [\widetilde{\mathfrak{a}}, \widetilde{\mathfrak{b}}] >$ from $\mathbb{L}_i^{\mathrm{r}}$;
  **if** $\widetilde{b}_{j,i} > \widetilde{\mathfrak{a}}$ **then**
    Add $< v, -1, [\widetilde{\mathfrak{a}}, \widetilde{b}_{j,i}] >$ to $\mathbb{L}_i^{\mathrm{r}}$;
  **if** $\widetilde{c}_{j,i} < \widetilde{\mathfrak{b}}$ **then**
    Add $< v, -1, [\widetilde{c}_{j,i}, \widetilde{\mathfrak{b}}] >$ to $\mathbb{L}_i^{\mathrm{r}}$;
  Add $< v, J_j, [\widetilde{b}_{j,i}, \widetilde{c}_{j,i}] >$ to $\mathbb{L}_{i,k}$;/* Suppose $v$ is the $k^{th}$ VM in $VC_i$ */
  $v_{j,i} \leftarrow v$;
**else**
  Search $\mathbb{L}_i^{\mathrm{o}}$ to find the first feasible idle time slot $[\widetilde{\mathfrak{a}}, \widetilde{\mathfrak{b}}]$ with $\max\{\widetilde{c}_{j,i-1} + \widetilde{d}_{j,i}, \widetilde{\mathfrak{a}}\} + \widetilde{p}_{j,i} \leq \min\{\widetilde{\delta}_{j,i}, \widetilde{\mathfrak{b}}\}$;
  **if** $< v, -1, [\widetilde{\mathfrak{a}}, \widetilde{\mathfrak{b}}] >$ *is found* **then**
    $\widetilde{b}_{j,i} \leftarrow \max\{\widetilde{c}_{j,i-1} + \widetilde{d}_{j,i}, \widetilde{\mathfrak{a}}\}$; $\widetilde{c}_{j,i} \leftarrow \widetilde{b}_{j,i} + \widetilde{p}_{j,i}$;
    Remove $< v, -1, [\widetilde{\mathfrak{a}}, \widetilde{\mathfrak{b}}] >$ from $\mathbb{L}_i^{\mathrm{o}}$;
    **if** $\widetilde{b}_{j,i} > \widetilde{\mathfrak{a}}$ **then**
      Add $< v, -1, [\widetilde{\mathfrak{a}}, \widetilde{b}_{j,i}] >$ to $\mathbb{L}_i^{\mathrm{o}}$;
    **if** $\widetilde{c}_{j,i} < \widetilde{\mathfrak{b}}$ **then**
      Add $< v, -1, [\widetilde{c}_{j,i}, \widetilde{\mathfrak{b}}] >$ to $\mathbb{L}_i^{\mathrm{o}}$;
    Add $< v, J_j, [\widetilde{b}_{j,i}, \widetilde{c}_{j,i}] >$ to $\mathbb{L}_{i,k}$;/* Suppose $v$ is the $k^{th}$ VM in $VC_i$ */
    $v_{j,i} \leftarrow v$;
  **else**
    Rent a new VM $v$;
    $\mathbb{O}_i \leftarrow \mathbb{O}_i \bigcup \{v\}$;
    $\widetilde{b}_{j,i} \leftarrow \widetilde{c}_{j,i-1} + \widetilde{d}_{j,i}$;
    Add $< v, -1, [t, \widetilde{b}_{j,i}] >$ to $\mathbb{L}_i^{\mathrm{o}}$;
    Add $< v, -1, [\widetilde{c}_{j,i}, \infty] >$ to $\mathbb{L}_i^{\mathrm{o}}$;
    Add $< v, J_j, [\widetilde{b}_{j,i}, \widetilde{c}_{j,i}] >$ to $\mathbb{L}_{i,k}$;/* Suppose $v$ is the $k^{th}$ VM in $VC_i$ */
    $v_{j,i} \leftarrow v$;
**return** $< \mathcal{T}_{j,i}, \widetilde{b}_{j,i}, \widetilde{c}_{j,i}, v_{j,i} >$;

is adjusted $mn_q$ times and $mn_q$ priorities are computed. Therefore, the complexity for the task prioritizing operation applying TP1 is $O(mn_q \log n_q)$

The time complexity of TP2 or TP3 is determined by the time for computing FEST of a task, which is equivalent to the time of searching for the first feasible time slot on the orthogonal list and it depends on the number of idle time slots in a virtual cluster. Suppose $n_i^k$ jobs have been assigned to the $k^{th}$ VM in $VC_i$ after $t$ which brings at most $n_i^k + 1$ idle fuzzy time slots on that VM. In other words, there are at most $\sum_{k=1}^{n_i^r+n_i^o}(n_i^k+1)$ down-linked nodes (idle time slots) on the orthogonal list of $VC_i$. $\sum_{k=1}^{n_i^r+n_i^o} n_i^k$ is actually the total number of unstarted tasks assigned to $VC_i$ after $t$. The number of currently scheduled and non-started tasks in all $m$ clusters is not more than $|\mathcal{F}^{\text{fuzzy}}(t)|$, which means the largest number of the down-linked nodes on the orthogonal list of one cluster is $\frac{|\mathcal{F}^{\text{fuzzy}}(t)|}{m}$, therefore the worst time complexity for computing the FEST is $O(\frac{|\mathcal{F}^{\text{fuzzy}}(t)|}{m})$. Actually, at most $m$ FEST values are computed in each iteration of FTS, because when the root task is arranged, the FEST values of the other tasks in $\mathcal{PQ}$ (at most $m$ ready tasks) have to be updated with the new availability of VMs. In total, the minimum heap is adjusted $mn_q$ times and $m^2n_q$ FEST values are computed. Therefore, the worst complexity for the task prioritizing operation in FTS with TP2/TP3 for EST is $O(mn_q \log n_q + mn_q|\mathcal{F}^{\text{fuzzy}}(t)|)$.

The time complexity of the task arranging operation is equivalent to the time complexity of finding the first feasible fuzzy time slot for the task, i.e., $O(\frac{|\mathcal{F}^{\text{fuzzy}}(t)|}{m})$. The task arranging operation is performed $mn_q$ times in FTS and the total time complexity of the task arranging operation is $O(n_q|\mathcal{F}^{\text{fuzzy}}(t)|)$.

Therefore, for the FTS with TP1, the time complexity is $O(mn_q log n_q + n_q \log |\mathcal{F}^{\text{fuzzy}}(t)|)$. For the FTS with TP2/TP3, the time complexity is $O(mn_q|\mathcal{F}^{\text{fuzzy}}(t)|)$.

## 5 Computational Experiments

In the proposed FDES framework there are several proposed variants for some of its components. We first calibrate these components and select the best combination for solving the considered problem. Then the calibrated algorithm is compared with the existing and highly related algorithms on effectiveness and robustness. All tested algorithms are coded in Java and run on an Intel Core $i7-4790$ CPU @$3.60GHz$ with 8 GBytes of RAM. All the evaluation is performed on the simulated clusters.

### 5.1 Parameter and Component Calibration

Since there are no comprehensive benchmarks available for the dynamic and fuzzy problem under study, we generate testing instances based on both existing studies and the cluster data collected from real cloud environments. We generate three types of workloads $W_1$, $W_2$ and $W_3$. Workloads of type $W_1$ and $W_2$ are constructed similarly to those for deterministic cloud scheduling problems in [6], [11], [31]. Workloads of type $W_3$ are constructed from the production data collected by Alibaba clusters [1].

Similar to [6], each test instance of type $W_1$ or $W_2$ includes 3600 JREs ($Q = 3600$) in an hour or one JRE per second ($\varepsilon = 1s$). The number of jobs $n_q$ in every JRE of a type $W_1$ instance follows a Poisson distribution $P(\lambda)$, $\lambda \in \{5, 10, \ldots, 50\}$. $n_q$ in every JRE of a type $W_2$ instance follows a Uniform distribution $U(0, 50)$. The Alibaba Cluster Trace "cluster-trace-v2017" provides a batch of workloads in 12 hours. Each record in the Trace includes the create time, the start time and the termination time of every task as well as its dependency with other tasks. We construct 12 $W_3$ workloads $W_{3,1}, \ldots, W_{3,12}$, i.e., $W_{3,i}$ $(i = 1, \ldots, 12)$ includes the real workload of the $i^{th}$ hour. Fig. 7 depicts the workload $W_{3,1}$. Though the average number of tasks per second is 192, there is no explicit probability distribution for task arrival.

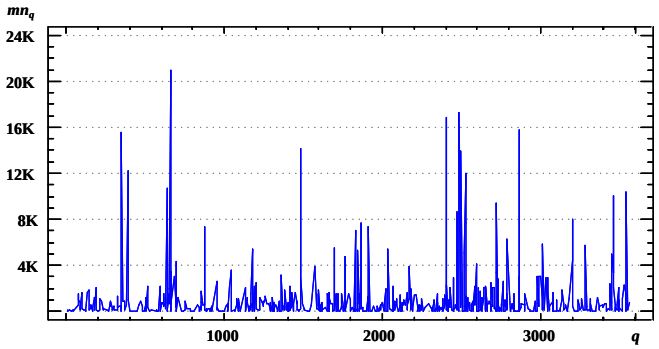


Fig. 7. One hour workload data from Alibaba Cluster Trace.

The temporal parameters for the fuzzy variables of a type $W_1$ or $W_2$ task $\mathcal{T}_{j,i}$ are set as follows:

- For fuzzy processing time $\widetilde{p}_{j,i}$, the most probable processing time $p_{j,i}^{\text{most}}$ is randomly generated in $[1, 10]$ (seconds), $p_{j,i}^{\min} = p_{j,i}^{\text{most}} \times (1 - e)$ and $p_{j,i}^{\max} = p_{j,i}^{\text{most}} \times (1 + e)$ where $e$ is a deviation factor. The real processing time $p_{j,i}$ is generated randomly following a normal distribution $N(p_{j,i}^{\text{most}}, (e \times p_{j,i}^{\text{most}})^2)$.
- Similarly, the fuzzy transmission time $\widetilde{d}_{j,i}$ is constructed by setting $d_{j,i}^{\text{most}}$ following a uniform distribution on a given interval $[a, b]$ based on the network measurements in [31], $d_{j,i}^{\min} = d_{j,i}^{\text{most}} \times (1 - e)$ and $d_{j,i}^{\max} = d_{j,i}^{\text{most}} \times (1 + e)$. The real transmission time $d_{j,i}$ is generated with a normal distribution $N(d_{j,i}^{\text{most}}, (e \times d_{j,i}^{\text{most}})^2)$.
- The fuzzy due date of $J_j$ is constructed by $D_j^1 = t_j + df_j \times \sum_{i=1}^{m} p_{j,i}^{\text{most}}$ and $D_j^2 = t_j + df_j \times \sum_{i=1}^{m} p_{j,i}^{\max}$ where $df_j$ is deadline factor following a normal distribution $N(df_{\text{avg}}, \sigma^2)$. We set $\sigma = 0.1 \times df_{\text{avg}}$ in this paper to try to avoid extreme due dates.

Different from the type $W_1$ and $W_2$ tasks, the temporal parameters for the fuzzy variables of a type $W_3$ task is set as follows: The real processing time $p_{j,i}$ is set as the real execution time provided in the Trace. The most probable processing time $p_{j,i}^{\text{most}}$ is generated randomly with a normal distribution $N(p_{j,i}, (e \times p_{j,i})^2)$. $p_{j,i}^{\min} = p_{j,i}^{\text{most}} \times (1 - e)$ and $p_{j,i}^{\max} = p_{j,i}^{\text{most}} \times (1 + e)$. Since there is no information related to the transmission time in the Trace, $\widetilde{d}_{j,i}$, $d_{j,i}$ and the fuzzy due date are set the same as the type $W_1$ or $W_2$ case.

The workload in each virtual cluster is the same on average since the processing times of tasks in different virtual clusters follows the same uniform distribution. Therefore, similar to [6], $n_i^r$ is initialized to be the same for each virtual cluster. Since it is very difficult to predict the number of reserved VMs $n_i^r$ precisely in advance, we just try to estimate it reasonably. $n_i^r$ is closely related to the job arrival rate $\lambda$, the number of tasks $m$ in each job and the deadline factor $df_{avg}$. Specifically, $n_i^r$ is proportional to $\lambda$ and $m$ while it is inversely proportional to $df_{avg}$. Based on [6], we propose a estimation model for $n_i^r$ as shown in Equ. (28) where the coefficient $\theta \in \{0.5, 1, 1.5\}$.

$$n_i^r = \frac{\lambda \times m \times \theta}{df_{\text{avg}}} \tag{28}$$

For the calibration instances with workload type $W_1$, there are $8 \times 4 \times 3 \times 3 \times 10 \times 3 = 8640$ instance combinations ($m \in \{3, 4, 5, 6, 7, 8, 9, 10\}$, $df_{\text{avg}} \in \{1.5, 2, 2.5, 3\}$, $d_{j,i}^{\text{most}} \sim U(10ms, 15ms), U(70ms, 80ms)$ or $U(250ms, 280ms)$, $e \in \{0.1, 0.2, 0.3\}$, $\lambda \in \{5, 10, \ldots, 50\}$ and $\theta \in \{0.5, 1, 1.5\}$).

For the calibration instances with workload type $W_2$, there are $8 \times 4 \times 3 \times 3 \times 3 = 864$ instance combinations ($m \in \{3, 4, 5, 6, 7, 8, 9, 10\}$, $df_{\text{avg}} \in \{1.5, 2, 2.5, 3\}$, $d_{j,i}^{\text{most}} \sim U(10ms, 15ms), U(70ms, 80ms)$ or $U(250ms, 280ms)$, $e \in \{0.1, 0.2, 0.3\}$ and $\theta \in \{0.5, 1, 1.5\}$), and 10 instances are randomly generated for each possible combination.

For the calibration instances with workload type $W_3$, there are $8 \times 4 \times 3 \times 3 \times 3 = 864$ instance combinations ($m \in \{3, 4, 5, 6, 7, 8, 9, 10\}$, $df_{\text{avg}} \in \{1.5, 2, 2.5, 3\}$, $d_{j,i}^{\text{most}} \sim U(10ms, 15ms), U(70ms, 80ms)$ or $U(250ms, 280ms)$, $e \in \{0.1, 0.2, 0.3\}$, $\theta \in \{0.5, 1, 1.5\}$), and 12 real one-hour workload data.

Therefore, the component variants of the proposal are calibrated on $8640 + 864 \times 10 + 864 \times 12 = 27648$ instances in total.

Solutions are evaluated by the LWS, and the lower bound for normalizing $C(\widetilde{T})$ is set to be the the best obtained value $C_{Best}$ (29). The cost on a reserved VM and an on-demand VM is set according to the real price setting at Amazon Elastic Compute Cloud: $\psi_i^r = 0.25$ and $\psi_i^o = 1$. Time unit $u$ for charging on-demand VMs is set to 600 seconds.

$$LWS = w \times \frac{C_{Best}}{C(\widetilde{T})} + (1 - w) \times S(\widetilde{T}) \tag{29}$$

In the FDES framework, there are two variants for the task deadline assignment (RTD and TTD), three variants for the task priority setting (TP1, TP2 and TP3) and two variants

1. https://github.com/alibaba/clusterdata

for the task collection strategy (TCS1 or TCS2). In other words, there are $2 \times 3 \times 2 = 12$ component combinations. Therefore, $12 \times 27648 = 331776$ 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 most $p$-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 were small in comparison.

Fig. 8 shows the LWS of component settings with $95.0\%$ Tukey Honest Significant Difference (HSD) intervals. It shows a statistically significant difference for the performance of different TP and TD settings on the LWS with $w \in \{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$. TP3 has better performance than TP1 and TP2, and TTD performs better than RTD. The performance of TCS1 and TCS2 is not statistically different. We make detailed comparison of these components with the different values of $w$. Fig. 9 depicts the interactions between $w$ and the compared components with 95.0% Tukey HSD intervals. It can be observed that TCS1 outperforms TCS2 if $w$ is bigger than approximately $0.5$, which indicates that in the optimized objective, if the satisfaction is more important than the cost, then TCS1 is the better option than TCS2. Otherwise, TCS2 can perform better than TCS1.

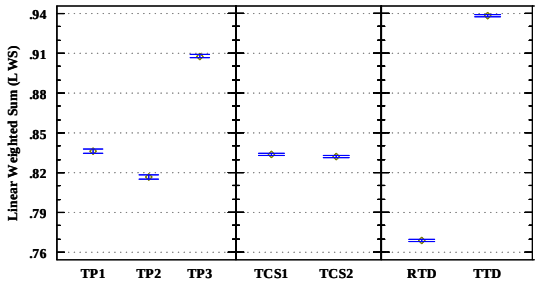


Fig. 8. The LWS of component settings with $95.0\%$ Tukey HSD intervals

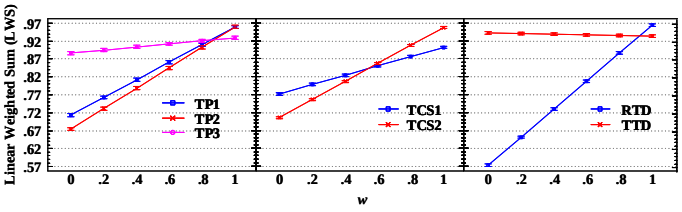


Fig. 9. The interactions between $w$ and the compared components with 95.0% Tukey HSD intervals

Therefore the best component combination when $w < 0.5$ is TP3 for TP, TTD for TD and TCS1 for TCS. If $w \geq 0.5$, the best one is TP3 for TP, TTD for TD and TCS2 for TCS.

### 5.2 Algorithm Comparison

Based on the above calibration, we set TP3 as TP, TTD as TD, TCS1 for TCS when $w < 0.5$ and TCS2 for TCS when $w \geq 0.5$ for the proposed FDES. The proposed FDES is compared with two existing algorithms HEFT [16] and DES [6] which were developed for the deterministic version of the considered problem. Both HEFT and DES are modified for operation on the fuzzy temporal parameters. The computation time of compared algorithms for scheduling one JRE is limited to 1000ms for a fair comparison.

We test instances with $W_1$, $W_2$ and $W_3$ workload types. 10 and 100 new instances are randomly generated for each of instance combinations of $W_1$ and $W_2$. For $W_3$ workload type instances, 12 real one-hour workload data are tested. Therefore, $8640 \times 10 + 864 \times 100 + 864 \times 12 = 183168$ new instances in total are used for comparison. LWSs of all instance combinations are shown in Tables 4 and 5.

Table 4 illustrates that for the problems with $W_1$ type workload, FDES obtains the highest LWSs with different $w$ settings (1.000, 0.973, 0.958, 0.930 and 0.908) except for $w = 1.0$. Whereas, DES has the lowest LWSs (0.478, 0.502, 0.626, 0.751) for $w < 0.8$ and the highest LWS (1.000) when $w = 1.0$. Table 5 illustrates that for the problems with $W_2$ type workload, FDES gives the highest LWSs with different $w$ settings (1.000, 0.975, 0.952, 0.930 and 0.909) except for $w = 1.0$. Whereas, DES has the lowest LWSs (0.372, 0.497, 0.623 and 748) for $w < 0.8$ and the highest LWS (1.000) when $w = 1.0$. Table 6 illustrates that for the problems with $W_3$ type workload, FDES gives the highest LWSs with different $w$ settings (0.996, 0.981, 0.966,0.952 and 0.940) except for $w = 1.0$. Whereas, HEFT has the lowest LWSs (0.619, 0.689, 0.760, 0.830 and 0.901) except for $w = 1.0$, and when $w = 1.0$ DES performs the best.

We find out that the local search heuristic DES, which is effective for the deterministic scheduling problems, does not perform well on the fuzzy scheduling problems. The reason may be that the local best fuzzy solutions obtained by a local search are very compact on the timetable, and there is little room for adjustments. When the deviation between the real solutions and the fuzzy solutions occurs, fuzzy and less compact solutions may gain an advantage. FDES can perform better than HEFT for most cases. The reasons may lie in that: (i) FDES adopts the more stringent deadline control strategy TTD and (ii) tasks may be rescheduled multiple times for optimizing the cost, while in HEFT each task is scheduled only once.

Figures 10-12 depict the interactions between instance parameters and the compared algorithms. From Fig. 10 we can observe that FDES is less sensitive to the parameter changes while the observed differences are statistically significant for DES and HEFT in most cases of the instances with $W_1$ type workload. $m$, $df_{avg}$, $e$ and $\theta$ have a large influence on the performance of DES and HEFT. Fig. 11 shows similar trends for the instances with $W_2$ type workload. The observed differences are statistically significant for DES and HEFT but not for FDES for most cases. Most of the parameters have a large influence on the performance of DES and HEFT. Fig. 12 shows similar trends for the instances with $W_3$ type workload. The observed differences are statistically significant for DES and HEFT whereas FDES

TABLE 4
Compared results on existing algorithms DES and HEFT and the proposed FDES on instances with $W_1$ type workload.

| Parm. | Value | $w = 0.0$ | | | $w = 0.2$ | | | $w = 0.4$ | | | $w = 0.6$ | | | $w = 0.8$ | | | $w = 1.0$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | DES | HEFT | FDES | DES | HEFT | FDES | DES | HEFT | FDES | DES | HEFT | FDES | DES | HEFT | FDES | DES | HEFT | FDES |
| $m$ | 3 | 0.254 | 0.662 | 0.995 | 0.236 | 0.725 | 0.987 | 0.427 | 0.788 | 0.980 | 0.618 | 0.852 | 0.973 | 0.808 | 0.915 | 0.967 | 0.999 | 0.978 | 0.965 |
| | 4 | 0.208 | 0.587 | 0.997 | 0.285 | 0.660 | 0.982 | 0.389 | 0.733 | 0.968 | 0.592 | 0.807 | 0.955 | 0.796 | 0.880 | 0.944 | 1.000 | 0.953 | 0.936 |
| | 5 | 0.210 | 0.541 | 1.000 | 0.368 | 0.617 | 0.976 | 0.526 | 0.693 | 0.953 | 0.684 | 0.768 | 0.930 | 0.841 | 0.844 | 0.909 | 0.999 | 0.920 | 0.890 |
| | 6 | 0.322 | 0.553 | 1.000 | 0.457 | 0.624 | 0.970 | 0.593 | 0.694 | 0.940 | 0.728 | 0.764 | 0.911 | 0.864 | 0.834 | 0.883 | 0.999 | 0.905 | 0.857 |
| | 7 | 0.513 | 0.595 | 1.000 | 0.610 | 0.657 | 0.972 | 0.707 | 0.720 | 0.939 | 0.804 | 0.783 | 0.907 | 0.902 | 0.846 | 0.876 | 0.999 | 0.909 | 0.846 |
| | 8 | 0.588 | 0.649 | 1.000 | 0.670 | 0.703 | 0.974 | 0.752 | 0.757 | 0.943 | 0.835 | 0.810 | 0.913 | 0.918 | 0.864 | 0.883 | 1.000 | 0.918 | 0.855 |
| | 9 | 0.662 | 0.687 | 1.000 | 0.729 | 0.736 | 0.984 | 0.797 | 0.785 | 0.955 | 0.864 | 0.835 | 0.925 | 0.931 | 0.884 | 0.897 | 0.999 | 0.934 | 0.869 |
| | 10 | 0.700 | 0.727 | 1.000 | 0.760 | 0.770 | 0.977 | 0.820 | 0.812 | 0.952 | 0.880 | 0.855 | 0.928 | 0.940 | 0.898 | 0.904 | 1.000 | 0.940 | 0.880 |
| $\lambda$ | 5 | 0.546 | 0.708 | 0.995 | 0.636 | 0.752 | 0.971 | 0.727 | 0.797 | 0.949 | 0.818 | 0.842 | 0.927 | 0.908 | 0.886 | 0.906 | 0.999 | 0.931 | 0.886 |
| | 10 | 0.474 | 0.675 | 0.999 | 0.579 | 0.726 | 0.975 | 0.684 | 0.778 | 0.951 | 0.789 | 0.829 | 0.929 | 0.894 | 0.880 | 0.907 | 0.999 | 0.932 | 0.886 |
| | 15 | 0.427 | 0.655 | 1.000 | 0.541 | 0.710 | 0.975 | 0.656 | 0.766 | 0.951 | 0.770 | 0.821 | 0.929 | 0.885 | 0.876 | 0.907 | 0.999 | 0.932 | 0.887 |
| | 20 | 0.394 | 0.636 | 1.000 | 0.514 | 0.696 | 0.977 | 0.636 | 0.755 | 0.953 | 0.757 | 0.814 | 0.930 | 0.878 | 0.873 | 0.908 | 0.999 | 0.933 | 0.888 |
| | 25 | 0.369 | 0.621 | 1.000 | 0.494 | 0.683 | 0.977 | 0.620 | 0.745 | 0.953 | 0.747 | 0.808 | 0.930 | 0.873 | 0.870 | 0.908 | 0.999 | 0.932 | 0.888 |
| | 30 | 0.348 | 0.608 | 1.000 | 0.478 | 0.673 | 0.977 | 0.608 | 0.738 | 0.953 | 0.738 | 0.802 | 0.930 | 0.869 | 0.867 | 0.908 | 0.999 | 0.932 | 0.888 |
| | 35 | 0.331 | 0.599 | 1.000 | 0.464 | 0.665 | 0.979 | 0.598 | 0.732 | 0.955 | 0.731 | 0.799 | 0.931 | 0.865 | 0.866 | 0.909 | 0.999 | 0.933 | 0.889 |
| | 40 | 0.312 | 0.590 | 1.000 | 0.450 | 0.658 | 0.980 | 0.587 | 0.727 | 0.956 | 0.725 | 0.796 | 0.932 | 0.862 | 0.864 | 0.909 | 0.999 | 0.933 | 0.889 |
| | 45 | 0.296 | 0.580 | 1.000 | 0.437 | 0.650 | 0.983 | 0.578 | 0.720 | 0.957 | 0.719 | 0.791 | 0.932 | 0.859 | 0.862 | 0.909 | 1.000 | 0.932 | 0.887 |
| | 50 | 0.282 | 0.579 | 1.000 | 0.425 | 0.650 | 0.986 | 0.569 | 0.721 | 0.960 | 0.713 | 0.791 | 0.934 | 0.856 | 0.862 | 0.910 | 1.000 | 0.933 | 0.889 |
| $df_{\text{avg}}$ | 1.5 | 0.150 | 0.740 | 1.000 | 0.320 | 0.778 | 0.995 | 0.490 | 0.816 | 0.978 | 0.660 | 0.855 | 0.962 | 0.830 | 0.894 | 0.946 | 1.000 | 0.932 | 0.931 |
| | 2 | 0.331 | 0.659 | 1.000 | 0.464 | 0.714 | 0.980 | 0.598 | 0.769 | 0.958 | 0.732 | 0.824 | 0.938 | 0.866 | 0.879 | 0.918 | 1.000 | 0.934 | 0.901 |
| | 2.5 | 0.465 | 0.576 | 1.000 | 0.572 | 0.647 | 0.972 | 0.679 | 0.718 | 0.944 | 0.785 | 0.789 | 0.918 | 0.892 | 0.860 | 0.893 | 0.999 | 0.934 | 0.870 |
| | 3 | 0.565 | 0.526 | 0.999 | 0.651 | 0.607 | 0.966 | 0.738 | 0.688 | 0.934 | 0.825 | 0.769 | 0.904 | 0.911 | 0.850 | 0.875 | 0.998 | 0.931 | 0.848 |
| $d_{j,i}^{\text{most}}$ | U(10,15) | 0.388 | 0.627 | 1.000 | 0.510 | 0.687 | 0.979 | 0.632 | 0.748 | 0.954 | 0.755 | 0.810 | 0.931 | 0.877 | 0.871 | 0.908 | 1.000 | 0.932 | 0.887 |
| | U(70,80) | 0.368 | 0.628 | 1.000 | 0.494 | 0.689 | 0.976 | 0.620 | 0.750 | 0.952 | 0.747 | 0.811 | 0.931 | 0.873 | 0.872 | 0.908 | 1.000 | 0.933 | 0.888 |
| | U(250,280) | 0.378 | 0.621 | 1.000 | 0.502 | 0.683 | 0.980 | 0.626 | 0.745 | 0.955 | 0.750 | 0.807 | 0.929 | 0.875 | 0.869 | 0.908 | 1.000 | 0.932 | 0.887 |
| $e$ | 0.1 | 0.487 | 0.883 | 1.000 | 0.589 | 0.893 | 0.977 | 0.692 | 0.904 | 0.954 | 0.794 | 0.915 | 0.931 | 0.897 | 0.925 | 0.909 | 0.999 | 0.936 | 0.887 |
| | 0.2 | 0.361 | 0.623 | 1.000 | 0.488 | 0.684 | 0.978 | 0.616 | 0.746 | 0.954 | 0.744 | 0.808 | 0.931 | 0.871 | 0.870 | 0.908 | 0.999 | 0.932 | 0.887 |
| | 0.3 | 0.287 | 0.369 | 1.000 | 0.429 | 0.481 | 0.978 | 0.571 | 0.539 | 0.953 | 0.714 | 0.705 | 0.930 | 0.857 | 0.817 | 0.907 | 0.999 | 0.929 | 0.888 |
| $\theta$ | 0.5 | 0.219 | 0.345 | 0.984 | 0.215 | 0.465 | 0.967 | 0.411 | 0.586 | 0.951 | 0.607 | 0.707 | 0.936 | 0.802 | 0.828 | 0.922 | 0.998 | 0.949 | 0.912 |
| | 1.0 | 0.447 | 0.685 | 1.000 | 0.557 | 0.730 | 0.973 | 0.668 | 0.775 | 0.940 | 0.778 | 0.820 | 0.909 | 0.888 | 0.865 | 0.879 | 0.999 | 0.910 | 0.850 |
| | 1.5 | 0.667 | 0.845 | 1.000 | 0.734 | 0.863 | 0.994 | 0.800 | 0.882 | 0.969 | 0.867 | 0.900 | 0.946 | 0.934 | 0.919 | 0.922 | 1.000 | 0.937 | 0.900 |
| Average | | 0.478 | 0.625 | 1.000 | 0.502 | 0.686 | 0.973 | 0.626 | 0.748 | 0.958 | 0.751 | 0.809 | 0.930 | 0.875 | 0.871 | 0.908 | 1.000 | 0.932 | 0.887 |

have the strongest robustness. Still, most of the parameters have a large influence on the performance of DES and HEFT.

It can be concluded, based on the above analysis, that the proposed FDES is the most effective and robust method among the compared algorithms, which implies that FDES is suitable for the considered problem.

## 6 Conclusions

The dynamic and fuzzy task scheduling problem on scalable resources in cloud platforms has been considered in this paper with the objectives of minimizing total rental costs and maximizing the users' satisfaction degree. The FDES framework has been proposed for the problem under study, which consists of a task collection component (TCS) and a fuzzy task scheduling component (FTS). TCS collects stochastic jobs and FTS schedules them periodically. In TCS, two task collection strategies TCS1 and TCS2 were presented, where the former schedules each task only once and the latter may schedule a task multiple times to reduce the cost. In the FTS, three strategies (TP1-3) were developed to compute priorities of ready tasks. In order to achieve fast search of available time slots on VMs, the two-dimensional orthogonal lists were employed to maintain the fuzzy time slots, based on which a task arrangement operation was introduced to generate the fuzzy assignments for a given task. We also defined the task deadline (TTD and RTD) strategies to constrain the latest completion times of tasks. A tight deadline setting (TTD) could lead to solutions with higher satisfaction degree than that of a relaxed deadline setting (RTD). By comparing with two existing and related algorithms (DES and HEFT), we have illustrated that FDES outperforms the compared algorithms statistically on both effectiveness and robustness.

## Acknowledgments

This work is supported by the National Key Research and Development Program of China (No. 2017YFB1400800), the National Natural Science Foundation of China (Nos. 61672297, 61872077, 61832004), the Natural Science Foundation of the Jiangsu Higher Education Institutions of China (Grant No. 18KJB520039) and the National Science Foundation for Post-doctoral Scientists of China (Grant No. 2018M640510). Rubén Ruiz is partially supported by the Spanish Ministry of Science, Innovation, and Universities,

TABLE 5
Compared results on existing algorithms DES and HEFT and the proposed FDES on instances with $W_2$ type workload.

| Parm. | Value | $w=0.0$ | | | $w=0.2$ | | | $w=0.4$ | | | $w=0.6$ | | | $w=0.8$ | | | $w=1.0$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | DES | HEFT | FDES | DES | HEFT | FDES | DES | HEFT | FDES | DES | HEFT | FDES | DES | HEFT | FDES | DES | HEFT | FDES |
| $m$ | 3 | 0.048 | 0.639 | 0.997 | 0.238 | 0.707 | 0.990 | 0.428 | 0.775 | 0.983 | 0.619 | 0.843 | 0.976 | 0.809 | 0.911 | 0.969 | 0.999 | 0.979 | 0.967 |
| | 4 | 0.109 | 0.565 | 1.000 | 0.287 | 0.642 | 0.984 | 0.465 | 0.719 | 0.970 | 0.643 | 0.796 | 0.955 | 0.822 | 0.873 | 0.943 | 1.000 | 0.951 | 0.934 |
| | 5 | 0.216 | 0.524 | 1.000 | 0.373 | 0.603 | 0.975 | 0.530 | 0.682 | 0.952 | 0.687 | 0.762 | 0.929 | 0.843 | 0.841 | 0.909 | 1.000 | 0.920 | 0.890 |
| | 6 | 0.333 | 0.546 | 1.000 | 0.466 | 0.618 | 0.969 | 0.600 | 0.690 | 0.939 | 0.733 | 0.762 | 0.911 | 0.867 | 0.834 | 0.884 | 1.000 | 0.906 | 0.858 |
| | 7 | 0.491 | 0.573 | 1.000 | 0.593 | 0.640 | 0.968 | 0.695 | 0.708 | 0.937 | 0.796 | 0.775 | 0.907 | 0.898 | 0.843 | 0.877 | 1.000 | 0.911 | 0.849 |
| | 8 | 0.578 | 0.621 | 1.000 | 0.662 | 0.681 | 0.969 | 0.746 | 0.741 | 0.940 | 0.831 | 0.801 | 0.911 | 0.915 | 0.860 | 0.883 | 1.000 | 0.920 | 0.856 |
| | 9 | 0.648 | 0.671 | 1.000 | 0.718 | 0.724 | 0.972 | 0.789 | 0.777 | 0.946 | 0.859 | 0.830 | 0.920 | 0.929 | 0.883 | 0.895 | 1.000 | 0.937 | 0.871 |
| | 10 | 0.674 | 0.700 | 1.000 | 0.739 | 0.749 | 0.976 | 0.804 | 0.799 | 0.952 | 0.869 | 0.849 | 0.929 | 0.934 | 0.898 | 0.908 | 0.999 | 0.948 | 0.886 |
| $df_{\text{avg}}$ | 1.5 | 0.146 | 0.718 | 0.999 | 0.317 | 0.760 | 0.985 | 0.488 | 0.802 | 0.970 | 0.658 | 0.845 | 0.956 | 0.829 | 0.887 | 0.942 | 1.000 | 0.930 | 0.929 |
| | 2 | 0.327 | 0.635 | 1.000 | 0.462 | 0.695 | 0.978 | 0.596 | 0.754 | 0.957 | 0.731 | 0.814 | 0.937 | 0.865 | 0.874 | 0.919 | 1.000 | 0.934 | 0.903 |
| | 2.5 | 0.461 | 0.547 | 1.000 | 0.569 | 0.624 | 0.972 | 0.677 | 0.701 | 0.945 | 0.784 | 0.779 | 0.919 | 0.892 | 0.856 | 0.895 | 1.000 | 0.933 | 0.873 |
| | 3 | 0.555 | 0.491 | 1.000 | 0.644 | 0.580 | 0.967 | 0.733 | 0.668 | 0.936 | 0.822 | 0.757 | 0.906 | 0.911 | 0.845 | 0.878 | 1.000 | 0.934 | 0.851 |
| $d_{j,i}^{\text{most}}$ | U(10,15) | 0.391 | 0.592 | 1.000 | 0.512 | 0.659 | 0.975 | 0.634 | 0.727 | 0.950 | 0.756 | 0.794 | 0.927 | 0.878 | 0.861 | 0.905 | 1.000 | 0.929 | 0.884 |
| | U(70,80) | 0.377 | 0.608 | 0.999 | 0.501 | 0.673 | 0.975 | 0.626 | 0.738 | 0.952 | 0.750 | 0.804 | 0.930 | 0.875 | 0.869 | 0.909 | 1.000 | 0.934 | 0.889 |
| | U(250,280) | 0.333 | 0.587 | 1.000 | 0.466 | 0.657 | 0.977 | 0.599 | 0.726 | 0.955 | 0.733 | 0.796 | 0.934 | 0.866 | 0.865 | 0.914 | 1.000 | 0.935 | 0.896 |
| $e$ | 0.1 | 0.467 | 0.874 | 1.001 | 0.573 | 0.886 | 0.977 | 0.680 | 0.899 | 0.954 | 0.787 | 0.912 | 0.932 | 0.893 | 0.924 | 0.911 | 1.000 | 0.937 | 0.890 |
| | 0.2 | 0.352 | 0.592 | 1.000 | 0.481 | 0.660 | 0.976 | 0.611 | 0.728 | 0.953 | 0.740 | 0.796 | 0.930 | 0.870 | 0.864 | 0.909 | 1.000 | 0.932 | 0.890 |
| | 0.3 | 0.282 | 0.321 | 0.998 | 0.425 | 0.443 | 0.974 | 0.569 | 0.564 | 0.951 | 0.712 | 0.686 | 0.928 | 0.856 | 0.808 | 0.908 | 1.000 | 0.929 | 0.890 |
| $\theta$ | 0.5 | 0.215 | 0.318 | 0.999 | 0.212 | 0.445 | 0.981 | 0.409 | 0.571 | 0.963 | 0.606 | 0.698 | 0.945 | 0.803 | 0.825 | 0.929 | 1.000 | 0.952 | 0.917 |
| | 1 | 0.427 | 0.654 | 1.000 | 0.541 | 0.706 | 0.968 | 0.656 | 0.758 | 0.939 | 0.771 | 0.810 | 0.910 | 0.885 | 0.861 | 0.882 | 1.000 | 0.913 | 0.856 |
| | 1.5 | 0.682 | 0.830 | 1.000 | 0.746 | 0.850 | 0.977 | 0.809 | 0.871 | 0.955 | 0.872 | 0.892 | 0.934 | 0.936 | 0.912 | 0.914 | 0.999 | 0.933 | 0.894 |
| Average | | 0.372 | 0.598 | 1.000 | 0.497 | 0.665 | 0.975 | 0.623 | 0.732 | 0.952 | 0.748 | 0.799 | 0.930 | 0.874 | 0.866 | 0.909 | 1.000 | 0.933 | 0.889 |

TABLE 6
Compared results on existing algorithms DES and HEFT and the proposed FDES on instances with $W_3$ type workload.

| Parm. | Value | $w=0.0$ | | | $w=0.2$ | | | $w=0.4$ | | | $w=0.6$ | | | $w=0.8$ | | | $w=1.0$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | DES | HEFT | FDES | DES | HEFT | FDES | DES | HEFT | FDES | DES | HEFT | FDES | DES | HEFT | FDES | DES | HEFT | FDES |
| $m$ | 3 | 0.583 | 0.577 | 0.991 | 0.666 | 0.656 | 0.977 | 0.748 | 0.734 | 0.963 | 0.831 | 0.812 | 0.950 | 0.913 | 0.891 | 0.937 | 0.996 | 0.969 | 0.927 |
| | 4 | 0.633 | 0.558 | 0.995 | 0.705 | 0.640 | 0.979 | 0.778 | 0.722 | 0.963 | 0.851 | 0.804 | 0.947 | 0.924 | 0.887 | 0.932 | 0.997 | 0.969 | 0.918 |
| | 5 | 0.616 | 0.588 | 0.998 | 0.692 | 0.664 | 0.982 | 0.768 | 0.740 | 0.966 | 0.844 | 0.816 | 0.950 | 0.919 | 0.893 | 0.935 | 0.995 | 0.969 | 0.922 |
| | 6 | 0.694 | 0.592 | 0.997 | 0.754 | 0.667 | 0.980 | 0.814 | 0.742 | 0.963 | 0.874 | 0.817 | 0.947 | 0.934 | 0.892 | 0.932 | 0.994 | 0.967 | 0.918 |
| | 7 | 0.700 | 0.616 | 0.999 | 0.759 | 0.687 | 0.984 | 0.817 | 0.759 | 0.968 | 0.876 | 0.830 | 0.954 | 0.934 | 0.902 | 0.941 | 0.993 | 0.973 | 0.929 |
| | 8 | 0.763 | 0.639 | 0.998 | 0.809 | 0.706 | 0.983 | 0.854 | 0.773 | 0.969 | 0.900 | 0.840 | 0.955 | 0.946 | 0.907 | 0.943 | 0.992 | 0.974 | 0.931 |
| | 9 | 0.756 | 0.698 | 1.000 | 0.803 | 0.753 | 0.986 | 0.851 | 0.809 | 0.974 | 0.898 | 0.864 | 0.962 | 0.945 | 0.920 | 0.951 | 0.993 | 0.975 | 0.941 |
| | 10 | 0.771 | 0.687 | 0.993 | 0.816 | 0.745 | 0.980 | 0.860 | 0.803 | 0.968 | 0.904 | 0.861 | 0.958 | 0.949 | 0.920 | 0.948 | 0.993 | 0.978 | 0.940 |
| $df_{\text{avg}}$ | 1.5 | 0.491 | 0.724 | 0.992 | 0.592 | 0.776 | 0.982 | 0.693 | 0.828 | 0.973 | 0.794 | 0.880 | 0.964 | 0.895 | 0.932 | 0.958 | 0.996 | 0.984 | 0.954 |
| | 2 | 0.701 | 0.692 | 0.997 | 0.759 | 0.749 | 0.983 | 0.818 | 0.806 | 0.969 | 0.877 | 0.863 | 0.956 | 0.936 | 0.920 | 0.944 | 0.994 | 0.977 | 0.933 |
| | 2.5 | 0.765 | 0.588 | 0.998 | 0.810 | 0.664 | 0.981 | 0.856 | 0.740 | 0.965 | 0.902 | 0.816 | 0.949 | 0.948 | 0.892 | 0.934 | 0.994 | 0.968 | 0.919 |
| | 3 | 0.802 | 0.474 | 0.998 | 0.840 | 0.571 | 0.979 | 0.878 | 0.668 | 0.961 | 0.917 | 0.765 | 0.942 | 0.955 | 0.862 | 0.924 | 0.993 | 0.959 | 0.907 |
| $d_{j,i}^{\text{most}}$ | U(10,15) | 0.528 | 0.442 | 0.995 | 0.621 | 0.546 | 0.980 | 0.713 | 0.651 | 0.964 | 0.806 | 0.755 | 0.950 | 0.899 | 0.860 | 0.936 | 0.992 | 0.964 | 0.924 |
| | U(70,80) | 0.713 | 0.633 | 0.998 | 0.769 | 0.701 | 0.981 | 0.825 | 0.768 | 0.965 | 0.881 | 0.835 | 0.950 | 0.937 | 0.903 | 0.935 | 0.993 | 0.970 | 0.921 |
| | U(250,280) | 0.827 | 0.783 | 0.996 | 0.861 | 0.823 | 0.983 | 0.896 | 0.862 | 0.971 | 0.930 | 0.902 | 0.959 | 0.964 | 0.941 | 0.949 | 0.998 | 0.981 | 0.939 |
| $e$ | 0.1 | 0.694 | 0.832 | 0.997 | 0.754 | 0.860 | 0.981 | 0.813 | 0.889 | 0.965 | 0.873 | 0.917 | 0.950 | 0.933 | 0.945 | 0.936 | 0.993 | 0.973 | 0.922 |
| | 0.2 | 0.693 | 0.623 | 0.997 | 0.754 | 0.692 | 0.982 | 0.814 | 0.762 | 0.968 | 0.874 | 0.832 | 0.953 | 0.934 | 0.901 | 0.940 | 0.994 | 0.971 | 0.928 |
| | 0.3 | 0.681 | 0.403 | 0.995 | 0.744 | 0.517 | 0.981 | 0.807 | 0.630 | 0.968 | 0.870 | 0.744 | 0.955 | 0.933 | 0.858 | 0.943 | 0.996 | 0.971 | 0.934 |
| $\theta$ | 0.5 | 0.583 | 0.460 | 0.997 | 0.665 | 0.561 | 0.979 | 0.747 | 0.662 | 0.961 | 0.830 | 0.763 | 0.944 | 0.912 | 0.928 | 0.864 | 0.994 | 0.965 | 0.913 |
| | 1 | 0.701 | 0.646 | 0.996 | 0.759 | 0.711 | 0.982 | 0.818 | 0.777 | 0.967 | 0.877 | 0.842 | 0.953 | 0.936 | 0.941 | 0.907 | 0.994 | 0.972 | 0.929 |
| | 1.5 | 0.785 | 0.753 | 0.996 | 0.827 | 0.798 | 0.983 | 0.868 | 0.843 | 0.972 | 0.910 | 0.888 | 0.961 | 0.952 | 0.951 | 0.933 | 0.994 | 0.977 | 0.942 |
| Average | | 0.689 | 0.619 | 0.996 | 0.750 | 0.689 | 0.981 | 0.811 | 0.760 | 0.966 | 0.872 | 0.830 | 0.952 | 0.933 | 0.901 | 0.940 | 0.994 | 0.971 | 0.928 |

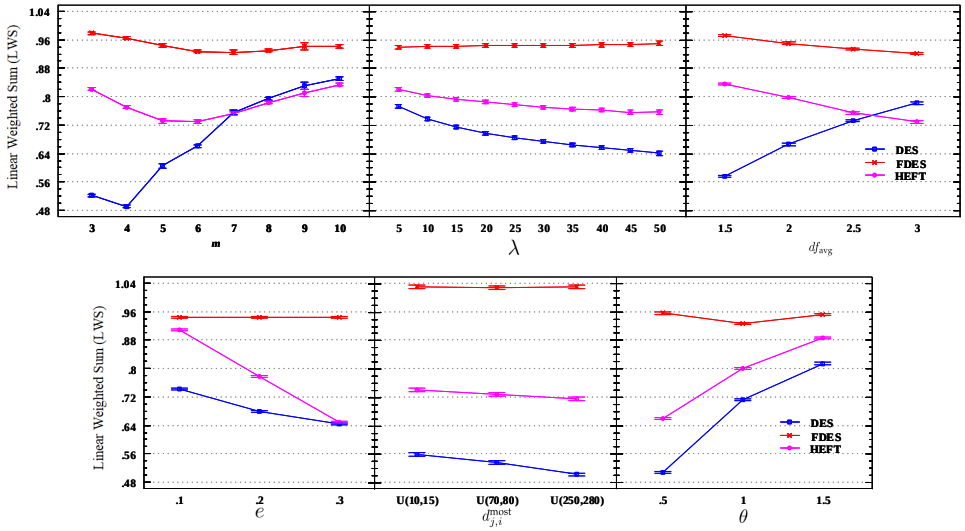


Fig. 10. Interactions between instance parameters and the compared algorithms with 95.0% Tukey HSD intervals for the instances with $W_1$ type workload

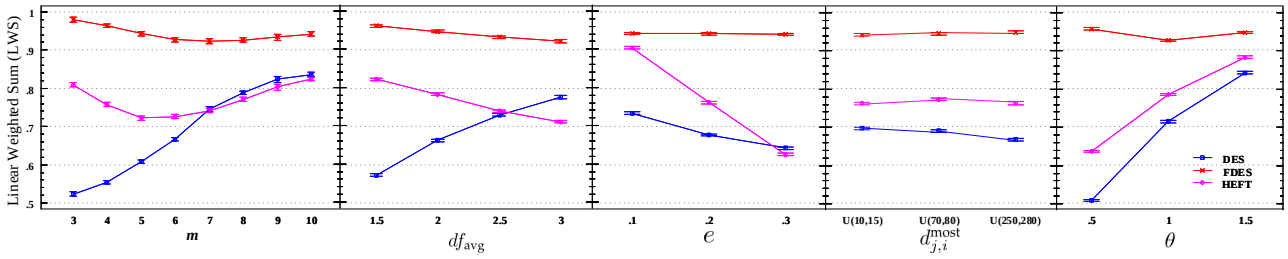


Fig. 11. Interactions between instance parameters and the compared algorithms with 95.0% Tukey HSD intervals for the instances with $W_2$ type workload

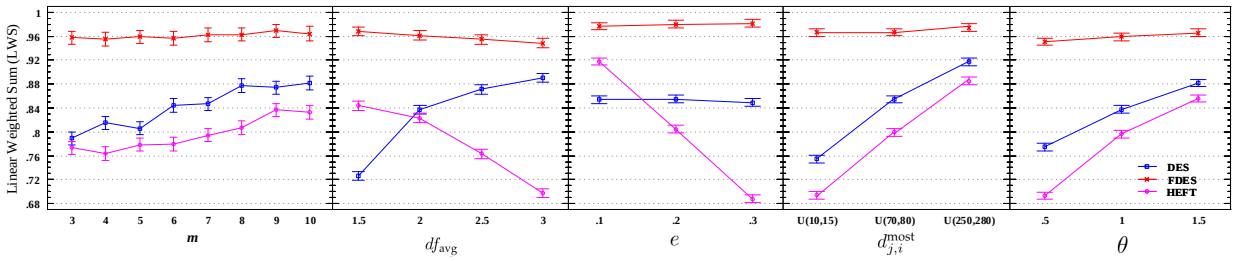


Fig. 12. Interactions between instance parameters and the compared algorithms with 95.0% Tukey HSD intervals for the instances with $W_3$ type workload

under the project "OPTEP-Port Terminal Operations Optimization" (No. RTI2018-094940-B-I00) financed with FEDER funds. The authors would like to thank the anonymous reviewers for their valuable feedback on this work.

## 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," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 2, pp. 363–373, 2014.

[2] X. Li, T. Jiang, and R. Ruiz, "Heuristics for periodical batch job scheduling in a Map-Reduce computing framework," *Information Sciences*, vol. 326, pp. 119–133, 2016.

[3] G. Muhammad, "Automatic speech recognition using interlaced derivative pattern for cloud based healthcare system," *Cluster Computing*, vol. 18, no. 2, pp. 1–8, 2015.

[4] Y. Xia, M. C. Zhou, X. Luo, Q. Zhu, J. Li, and Y. Huang, "Stochastic modeling and quality evaluation of infrastructure-as-a-service clouds," *IEEE Transactions on Automation Science & Engineering*, vol. 12, no. 1, pp. 162–170, 2014.

[5] 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.

[6] J. Zhu, X. Li, R. Ruiz, and X. Xu, "Scheduling stochastic multi-stage jobs to elastic hybrid cloud resources," *IEEE Transactions on Parallel & Distributed Systems*, vol. 29, no. 6, pp. 1401–1415, 2018.

[7] 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, pp. 61–78, 2012.
[8] 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.
[9] E. Gelenbe and S. Timotheou, "Random neural networks with synchronized interactions." *Neural Computation*, vol. 20, no. 9, pp. 2308–24, 2008.
[10] L. Wang and E. Gelenbe, "Adaptive dispatching of tasks in the cloud," *IEEE Transactions on Cloud Computing*, vol. 6, no. 1, pp. 33–45, 2018.
[11] Z. Wang, M. M. Hayat, N. Ghani, and K. B. Shaban, "Optimizing cloud-service performance: Efficient resource provisioning via optimal workload allocation," *IEEE Transactions on Parallel & Distributed Systems*, vol. 28, no. 6, pp. 1689–1702, 2017.
[12] 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.
[13] 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.
[14] J. Taheri, A. Y. Zomaya, H. J. Siegel, and Z. Tari, "Pareto frontier for job execution and data transfer time in hybrid clouds," *Future Generation Computer Systems*, vol. 37, no. 7, pp. 321–334, 2014.
[15] H. Yuan, J. Bi, W. Tan, M. Zhou, B. H. Li, and J. Li, "Ttsa: An effective scheduling approach for delay bounded tasks in hybrid clouds," *IEEE Transactions on Cybernetics*, vol. 47, no. 11, pp. 3658–3668, 2017.
[16] 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.
[17] E. D. Coninck, T. Verbelen, B. Vankeirsbilck, S. Bohez, P. Simoens, and B. Dhoedt, "Dynamic auto-scaling and scheduling of deadline constrained service workloads on iaas clouds," *Journal of Systems & Software*, vol. 118, pp. 101–114, 2016.
[18] R. Duan, R. Prodan, and X. Li, "Multi-objective game theoretic scheduling of bag-of-tasks workflows on hybrid clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 1, pp. 29–42, 2014.
[19] S. Li, Y. Zhou, L. Jiao, X. Yan, X. Wang, and R. T. Lyu, "Towards operational cost minimization in hybrid clouds for dynamic resource provisioning with delay-aware optimization," *IEEE Transactions on Services Computing*, vol. 8, no. 3, pp. 398–409, 2015.
[20] L. F. Bittencourt, E. R. M. Madeira, and N. L. S. D. Fonseca, "Scheduling in hybrid clouds," *Communications Magazine IEEE*, vol. 50, no. 9, pp. 42–47, 2012.
[21] H. Mohammadi Fard, R. Prodan, and T. Fahringer, "A truthful dynamic workflow scheduling mechanism for commercial multi-cloud environments," *IEEE Transactions on Parallel & Distributed Systems*, vol. 24, no. 6, pp. 1203–1212, 2013.
[22] P. Lu, Q. Sun, K. Wu, and Z. Zhu, "Distributed online hybrid cloud management for profit-driven multimedia cloud computing," *IEEE Transactions on Multimedia*, vol. 17, no. 8, pp. 1297–1308, 2015.
[23] S. Chanas and A. Kasperski, "On two single machine scheduling problems with fuzzy processing times and fuzzy due dates," *European Journal of Operational Research*, vol. 147, no. 2, pp. 281–296, 2003.
[24] S. Balin, "Parallel machine scheduling with fuzzy processing times using a robust genetic algorithm and simulation," *Information Sciences*, vol. 181, no. 17, pp. 3551–3569, 2011.
[25] W. C. Yeh, P. J. Lai, W. C. Lee, and M. C. Chuang, "Parallel-machine scheduling to minimize makespan with fuzzy processing times and learning effects," *Information Sciences*, vol. 269, no. 4, pp. 142–158, 2014.
[26] W. Huang, S. K. Oh, and W. Pedrycz, "A fuzzy time-dependent project scheduling problem," *Information Sciences*, vol. 246, no. 14, pp. 100–114, 2013.
[27] M. Sakawa and R. Kubota, "Fuzzy programming for multi-objective job shop scheduling with fuzzy processing time and fuzzy duedate through genetic algorithms," *European Journal of Operational Research*, vol. 120, no. 2, pp. 393–407, 2000.
[28] S. Abdullah and M. Abdolrazzagh-Nezhad, "Fuzzy job-shop scheduling problems: A review," *Information Sciences*, vol. 278, pp. 380–407, 2014.
[29] 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.
[30] S. Wang, G. L. Aori, G. Liu, and S. Gao, "A hybrid discrete imperialist competition algorithm for fuzzy job-shop scheduling problems," *IEEE Access*, vol. 4, pp. 9320–9331, 2016.
[31] L. Wang, O. Brun, and E. Gelenbe, "Adaptive workload distribution for local and remote clouds," in *IEEE International Conference on Systems, Man, and Cybernetics*, 2016, Budapest, Hungary, pp. 3984–3988.

**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 Science. 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** (Senior Member, IEEE) 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, and the Ph.D. degree in Applied Computer Science from the Harbin Institute of Technology, Harbin, China, in 2002. He is currently a distinguished professor at the School of Computer Science and Engineering, Southeast University, Nanjing, China. He is the author or co-author over more than 100 academic papers, some of which have been published in international journals such as *IEEE Transactions on Computers*; *IEEE Transactions on Parallel and Distributed Systems*; *IEEE Transactions on Services Computing*; *IEEE Transactions on Cybernetics*; *IEEE Transactions on Automation Science and Engineering*; *IEEE Transactions on Cloud Computing*; *IEEE Transactions on Systems, Man and Cybernetics: Systems*; *Information Sciences*; *Omega* and *European Journal of Operational Research*. His research interests include Scheduling in Cloud Computing, Scheduling in Cloud Manufacturing, Service Computing, Big Data and Machine Learning.

**Rubén Ruiz** is a full professor of Statistics and Operations Research at the Universitat Politècnica de València, Spain. He is co-author of more than 80 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* 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 Universitat Politècnica de València 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.

**Wei Li** (SM'16) received his PhD degree from School of Information Technologies at The University of Sydney. He is currently a research associate in Centre for Distributed and High Performance Computing, School of Computer Science, The University of Sydney. His research interests include Internet of Things, edge computing, sustainable computing, task scheduling, energy efficiency and optimization. He is the recipient of four IEEE or ACM conference best paper awards. He received the IEEE TCSC Award for Excellence in Scalable Computing for Early Career Researchers (2018) and the IEEE Outstanding Leadership Award (2018). He is a senior member of the IEEE Computer Society and the IEEE, and a member of the ACM.

**Haiping Huang** (M'07) received the B.Eng. and M.Eng. degrees in computer science and technology from the Nanjing University of Posts & Telecommunications, Nanjing, China, in 2002 and 2005, respectively, and the Ph.D. degree in computer application technology from Soochow University, Suzhou, China, in 2009. From May 2013 to November 2013, he was a Visiting Scholar with the School of Electronics and Computer Science, University of Southampton, Southampton, U.K. He is currently a Professor with the School of Computer Science and Technology, Nanjing University of Posts & Telecommunications. His research interests include information security and privacy protection of wireless sensor networks.

**Albert Y. Zomaya** (F'04) is the Chair Professor of high performance computing and networking with the School of Computer Science, The University of Sydney, Australia, and he also serves as the Director with the Centre for Distributed and High Performance Computing. He has authored and coauthored more than 600 scientific papers and articles and is author, co-author, or Editor of more than 20 books. He is the Founding Editor in Chief of the IEEE TRANSACTIONS ON SUSTAINABLE COMPUTING and serves as an Associate Editor for more than 20 leading journals. He served as an Editor-in-Chief for the IEEE TRANSACTIONS ON COMPUTERS (2011-2014).

Prof. Zomaya was the recipient of the IEEE Technical Committee on Parallel Processing Outstanding Service Award (2011), the IEEE Technical Committee on Scalable Computing Medal for Excellence in Scalable Computing (2011), and the IEEE Computer Society Technical Achievement Award (2014). He is a Chartered Engineer, a member of Academia Europaea, a fellow of AAAS, IEEE, and IET. His research interests include the areas of parallel and distributed computing and complex systems.