Future Generation Computer Systems 93 (2019) 278–289

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based HEFT ☆

Xiumin Zhou [a], Gongxuan Zhang [a], Jin Sun [a], Junlong Zhou [a,*], Tongquan Wei [b], Shiyan Hu [c]

[a] School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, 210094, China
[b] Department of Computer Science and Technology, East China Normal University, Shanghai 200241, China
[c] Department of Electrical and Computer Engineering, Michigan Technological University, MI 49931, USA

HIGHLIGHTS

- Achieving the joint optimization of monetary cost and makespan.
- Featuring the consideration of real-world pricing and resource models.
- Designing a novel workflow scheduling algorithm.
- Conducting a full validation on both real-world and synthetic workflows.

ARTICLE INFO

Article history:
Received 11 June 2018
Received in revised form 20 October 2018
Accepted 24 October 2018
Available online 3 November 2018

Keywords:
Cloud computing
Workflow scheduling
Multi-objective optimization
Fuzzy dominance sort
HEFT

ABSTRACT

More and more enterprises and communities choose cloud computing platforms to deploy their commercial or scientific workflow applications along with the increasing popularity of pay-as-you-go cloud services. A major task of cloud service providers is to minimize the monetary cost and makespan of executing workflows in the Infrastructure as a Service (IaaS) cloud. Most of the existing techniques for cost and makespan minimization are designed for traditional computing platforms which cannot be applied to the cloud computing platforms with unique service-based resource managing methods and pricing strategies. In this paper, we study the joint optimization of cost and makespan of scheduling workflows in IaaS clouds, and propose a novel workflow scheduling scheme. In this scheme, a fuzzy dominance sort based heterogeneous earliest-finish-time (FDHEFT) algorithm is developed which closely integrates the fuzzy dominance sort mechanism with the list scheduling heuristic HEFT. Extensive experiments using the real-world and synthetic workflows demonstrate the efficacy of our scheme. Our scheme can achieve significantly better cost-makespan tradeoff fronts with remarkably higher Hypervolume and can run up to hundreds of times faster than the state-of-the-art algorithms.

1. Introduction

Cloud computing has become a very popular and effective commercial computing model that distributes user requests on a shared resource pool and delivers hosted services over Internet. As a business model, it turns computing, storage, and communication resources into ordinary commodities and utilities in a pay-as-you-go manner [1–3]. This feature brings the opportunities of large-scale computation without physically owning a cloud. Infrastructure as a Service (IaaS) is one of the most common cloud service models, offering users the ability to provision or release pre-configured virtual machines (VMs) from a cloud infrastructure. Using the VMs, users can access to almost unlimited number of computing resources with much lower ownership cost for executing applications [4].

Workflow is a widely-used model to describe scientific and data-intensive applications deployed and hosted on the IaaS clouds such as Amazon EC2 and other cloud providers [5–7]. It is formed by a number of tasks and the data or control dependencies between tasks and can be represented as a directed acyclic graph where nodes represent tasks and edges represent data or control dependencies. Users buy services from the IaaS service provider to execute their submitted workflows, each of which is usually associated with a deadline for the quality of services (QoS) requirement. The IaaS service provider charges users based on the execution of their workflows and QoS requirements. Therefore, it is natural for the service provider to pursue the goal of reducing monetary

☆ This work was partially supported by National Natural Science Foundation of China under Grants 61802185, 61272420, 61872185, and 61502234, Natural Science Foundation of Jiangsu Province, China under Grants BK20180470 and BK20150785, and Natural Science Foundation of Shanghai, China under Grant 16ZR1409000.

* Corresponding author.
E-mail address: jlzhou@njust.edu.cn (J. Zhou).

https://doi.org/10.1016/j.future.2018.10.046
0167-739X/

cost and execution makespan, in order to gain more profits and ensure QoS. Alkhanak et al. [8] summarized the challenges in cost-aware approaches considering QoS performance (e.g., makespan) as well as system functionality and architecture. Following the concerns discussed in [8], in this paper we focus on designing a workflow scheduling scheme that minimizes both monetary cost and makespan.

Considerable research efforts have been devoted to the investigation of workflow scheduling for optimizing monetary cost and execution makespan in heterogeneous computing environments. Multi-objective optimization are the most common approaches used to minimize monetary cost and execution makespan simultaneously. Su et al. [9] formulated an optimization problem of scheduling tasks with multiple VMs and different pricing models for jointly minimizing cost and makespan, and proposed a pareto optimal scheduling heuristic that combines cost and makespan into one parameter to represent the preference of the both objectives. Based on the classical Particle Swarm Optimization (PSO) algorithm, Garg and Singh developed two improved versions of PSO, i.e., non-dominated sort PSO (NSPSO) [10] and ε-Fuzzy PSO [11]. NSPSO [10] extends the basic form of PSO by making a better use of particle personal bests and offspring for effective non-domination comparisons while ε-Fuzzy PSO [11] incorporates a fuzzy based mechanism into PSO for determining the best compromised solutions. Similarly, NSGAII* and SPEA2* [12] are designed for the workflow scheduling problem, which improve the evolutionary algorithms NSGAII and SPEA2, respectively. Recently, Durillo et al. [13] introduced a novel multi-objective method called multi-objective heterogeneous earliest-finish-time (MOHEFT) algorithm for scheduling workflows in Amazon EC2, which is an extension to the well-known list heuristic HEFT [14] that solves the mono-objective workflow scheduling problem. Unlike HEFT [14], MOHEFT [13] builds several intermediate workflow schedules in parallel in each step instead of a single schedule and uses dominance relationships and crowding distance to ensure the diversity of tradeoff solutions. However, all the above works cannot be directly applied to cloud environments since they are almost designed for traditional heterogeneous computing environments.

In this paper, we focus on the workflow scheduling problem of minimizing cost and makespan simultaneously under the precedence constraints of tasks in the workflow. We design a fuzzy dominance sort based heterogeneous earliest-finish-time (FDHEFT) algorithm to solve the workflow scheduling problem in cloud. Similar to MOHEFT, FDHEFT is also an improved version of HEFT and can be divided into two major phases, i.e., task prioritizing phase and instance selection phase. In the task prioritizing phase, the scheduling priorities of all tasks in the workflow are assigned and then in the instance selection phase, the best instance for each task in the scheduling list is determined. Compared to MOHEFT, FDHEFT can not only achieve a lower time overhead by pruning the candidate tradeoff solutions but also find out the better solutions by using fuzzy dominance sort. To the best of our knowledge, FDHEFT is the first attempt that extends the list-based heuristic HEFT for multi-objective workflow scheduling in a cloud computing environment using fuzzy dominance sort. The major contributions of this paper can be summarized as follows.

- We formulate the problem of jointly minimizing monetary cost and makespan for executing workflows in cloud under the constraint of task precedence.
- We propose a new list scheduling algorithm FDHEFT to solve the multi-objective workflow scheduling problem. Specifically, we apply fuzzy dominance sort to HEFT and use fuzzy dominance to measure the relative fitness of solutions in multi-objective domain.
- We conduct extensive simulation experiments to validate FDHEFT. We compare our FDHEFT with a number of representative multi-objective optimization algorithms, including NSPSO [10], ε-Fuzzy PSO [11], SPEA2* [12], and MOHEFT [13]. Extensive experimental results on standard and synthetic workflow applications show the efficacy of our scheme.

Our scheme can achieve better solutions with higher Hypervolume and less CPU runtime as compared to a number of peer approaches. Concretely, the cost-makespan pareto optimal solutions obtained by our algorithms have a distinct advantage over the peer approaches and the time overhead of our algorithms for generating solutions are only a small percentage of that of the peer approaches.

The remainder of this paper is organized as follows. Section 2 introduces system models and Section 3 defines our concerned workflow scheduling problem. Section 4 describes our proposed FDHEFT algorithm to solve the problem and Section 5 validates the effectiveness of FDHEFT. Concluding remarks are given in Section 6.

2. System models

This section introduces the workflow model as well as the cloud resource model used in the paper.

2.1. Workflow model

The structure of a workflow W can be modeled as a direct acyclic graph (DAG) $G = (T, D)$, where $T = \{T_1, T_2, \ldots, T_n\}$ represents the set of n tasks in the workflow application and $D = \{(T_i, T_j)|T_i, T_j \in T\}$ represents the set of data flow dependencies among tasks. The data flow dependency, represented by (T_i, T_j), indicates that there is a precedence constraint between tasks T_i and T_j, where task T_i is an immediate predecessor of task T_j and task T_j is an immediate successor of task T_i. Since a task may have multiple predecessors and successors, we use $\mathrm{Pre}(T_i)$ and $\mathrm{Succ}(T_i)$ to denote the set of immediate predecessors and successors of task T_i. That is,

$$\mathrm{Pre}(T_i) = \{T_j|(T_j, T_i) \in D\}, \tag{1}$$

$$\mathrm{Succ}(T_i) = \{T_j|(T_i, T_j) \in D\}. \tag{2}$$

A task without any predecessors is called an entry task T_{entry} and satisfies

$$\mathrm{Pre}(T_{\mathrm{entry}}) = \emptyset, \tag{3}$$

while a task without any successors is called an exit task T_{exit} and satisfies

$$\mathrm{Succ}(T_{\mathrm{exit}}) = \emptyset. \tag{4}$$

Fig. 1 gives an example of a workflow represented by a DAG with 7 nodes and 10 edges. As we know, most workflow scheduling algorithms require a DAG with only one T_{entry} and one T_{exit}, which can be easily realized by adding a pseudo T_{entry} and/or a pseudo T_{exit} with zero weight to the DAG. As in [4,15,16], we also use the same assumption that every workflow has only one T_{entry} and one T_{exit}.

2.2. Resource model

IaaS provides pre-configured VMs from a cloud infrastructure for users to deploy their own applications, and thus are most suitable for executing workflows [17]. The running VM in an IaaS platform is also called an instance. As we know, an IaaS platform is able to provide a variety of instance types such as CPU capacity, network bandwidth, and memory storage. For example, real-world

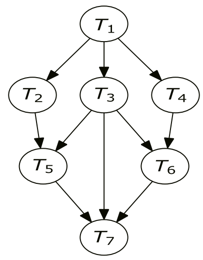


Fig. 1. An example of workflow application with data precedence.

IaaS cloud platform Amazon EC2 provides VM instances with different CPU capacities to meet different demands of various applications. In this paper, we consider CPU capacities that determine task execution time and bandwidths that affect data transformation time, for each instance type.

Let $I = \bigcup_{s=1}^{\infty}\{I_s\}$ denote all the available instances in an IaaS platform, which means the number of instances that a client can access is unlimited. Let $R = \bigcup_{k=1}^{m}\{R_k\}$ denote the all the instance types where m is the number of the types. Then we can use $R_k = \text{Type}(I_s)$ to indicate the relation that instance I_s belongs to type R_k. The concept of compute unit (CU) is adopted by IaaS providers to accurately describe the CPU capacities of different instance types, which follows that the larger the CU, the higher computing performance of the instance. Apparently, CU determines the execution time of tasks. For example, if the CU of an instance is doubled, the execution time of the tasks running on this instance would be halved. We also define that the reference execution time of a task is the time of executing this task on an instance whose CU equals to 1. Let $\text{CU}(T_i)$ and $\text{RET}(T_i)$ be the compute unit and reference execution time of task T_i, respectively, the actual execution time $\text{AET}(T_i)$ of task T_i running on instance I_s is then formulated as

$$\text{AET}(T_i) = \frac{\text{RET}(T_i)}{\text{CU}\big(\text{Type}(I_s)\big)} = \frac{\text{RET}(T_i)}{\text{CU}\big(R_k\big)}. \tag{5}$$

$$\text{CMT}(T_i, T_j) = \begin{cases} \frac{\text{DATA}(T_i, T_j)}{\min\left\{\text{BW}\big(\text{Type}(I_\alpha)\big), \text{BW}\big(\text{Type}(I_\beta)\big)\right\}}, & I_\alpha \neq I_\beta, \\ 0, & I_\alpha = I_\beta. \end{cases} \tag{6}$$

Inter-task communications need to be taken into account due to the data flow dependency among tasks. The communication time between two tasks is determined by the communication bandwidths of instances and the size of transferred data, and is considered negligible only when the two tasks are running on the same instance. In general, the communication bandwidths are different for different instance types. The order of data transfers among tasks on instances is consistent with the scheduling order of tasks, which is decided by a rank function and is discussed in detail later. Taking tasks $T_1 - T_3$ of Fig. 1 as an example, we assume that task T_1 is running on instance I_1 and tasks T_2, T_3 are running on instance I_2. If the scheduling order of these tasks is determined as $\langle T_1, T_2, T_3\rangle$, the data transfer between T_1 and T_3 accordingly needs to be after the data transfer between T_1 and T_2. In other words, the two data transfers would not use the same communication link at the same time. Thus we can safely assume that each data transfer is allowed to use the full bandwidth, same as in [4]. Let $\text{BW}(R_k)$ represent the bandwidth of instance type R_k, and $\text{DATA}(T_i, T_j)$ represent the size of data transferred from T_i to T_j. Supposing tasks T_i and T_j are running on instances I_α and I_β, respectively, the communication time $\text{CMT}(T_i, T_j)$ between tasks T_i and T_j is then described in Eq. (6).

3. Problem definition

This section first formulates the multi-objective workflow scheduling problem that we are trying to solve and then introduces the basics of multi-objective optimization.

3.1. Problem formulation

In this paper, we address the scheduling problem of minimizing makespan and cost simultaneously for workflows in cloud. Before giving the formal definition of our problem, we first show how to calculate makespan and cost. Let $\text{ST}(T_i)$ and $\text{FT}(T_i)$ denote the start time and finish time of task T_i, respectively. Since the start time of task T_i depends on the finish time of all its predecessors $\text{Pre}(T_i)$, the communication time $\text{CMT}(T_j, T_i)$ between its predecessors and itself, and the finish time $\text{FT}(T_j)$ of the previous task that has been executed on the same instance, the finish time $\text{FT}(T_i)$ of task T_i is calculated as

$$\text{FT}(T_i) = \text{ST}(T_i) + \text{AET}(T_i) = \max\Big\{\text{AVA}\big(\text{Ins}(T_i)\big), \max_{T_j \in \text{Pre}(T_i)}\big(\text{FT}(T_j) + \text{CMT}(T_j, T_i)\big)\Big\} + \text{AET}(T_i), \tag{7}$$

where $\text{AVA}\big(\text{Ins}(T_i)\big)$ is the available time of instance $\text{Ins}(T_i)$ that task T_i executes on, and changes dynamically during scheduling. Note that the start time of entry task T_{entry} is zero, i.e., $\text{FT}(T_{\text{entry}}) = 0$. The makespan defined as the finish time of exit task T_{exit} is then formulated as

$$\text{Makespan} = \text{FT}(T_{\text{exit}}), \tag{8}$$

where $\text{FT}(T_{\text{exit}})$ can be obtained using Eq. (7).

Let $PC = \{PC_1, PC_2, \ldots, PC_H\}$ denote the set of pricing options for using the services provided by an IaaS platform, and $\text{MC}(PC_h, R_k, I_s)$ denote the monetary cost of running instance I_s with type R_k using pricing model PC_h, where $PC_h \in PC$ $(1 \le h \le H)$. We do not have any further assumption on the pricing model such that the model could be generic for most IaaS platforms. Based on the pricing model, the total monetary cost of executing all tasks in the workflow can be expressed as

$$\begin{aligned} \text{Cost} &= \sum_{I_s \in I^*} \text{MC}(PC_h, \text{Type}(I_s), I_s) \\ &= \sum_{I_s \in I^*} \text{MC}(PC_h, R_k, I_s), \end{aligned} \tag{9}$$

where $I^* = \{I_s \mid \exists T_i \in T : \text{Ins}(T_i) = I_s\}$ is the set of instances used to execute all tasks in the set T.

Now we define our studied problem as follows. Given a workflow represented by a directed acyclic graph $G = (T, D)$ and an IaaS platform represented by $S = (I, R, PC)$, design a workflow scheduling scheme that determines the scheduling order of tasks and two assignments (task-to-instance assignment and instance-to-type assignment), in order to minimize makespan (given in Eq. (8)) and cost (given in Eq. (9)) simultaneously while satisfying the constraint on task precedence. The task scheduling order is used to ensure the dependency constraints between tasks, that is, a task cannot be scheduled unless all its predecessors have been scheduled. The task-to-instance assignment indicates which instance every task is put on while the instance-to-type assignment indicates which type of every instance is.

3.2. Basics of multi-objective optimization

Our workflow scheduling problem of jointly minimizing makespan and cost is a typical multi-objective optimization problem (MOP), which is a type of problem that has several conflicting

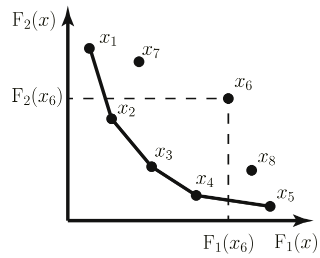


Fig. 2. Pareto optimal front for bi-objective minimization.

objectives which need to be optimized simultaneously:

$$\begin{aligned}&\text{Minimize } \mathbf{F}(x) = \big(\mathbf{F}_1(x), \mathbf{F}_2(x), \ldots, \mathbf{F}_r(x)\big)^T\\&\text{Subject to } \ x \in X\end{aligned} \tag{10}$$

where X is the solution space and $\mathbf{F}_r(x)$ is the rth objective of solution x. Our concerned workflow scheduling problem is an MOP since optimizing makespan and cost conflict with each other. To handle the conflict between both objectives, pareto dominance is commonly used to compare solutions. Let x_1 and x_2 be the two solutions to an MOP (i.e., $x_1, x_2 \in X$), x_1 is then said to dominate x_2 if and only if

$$\forall p : \mathbf{F}_p(x_1) \leq \mathbf{F}_p(x_2) \wedge \exists q : \mathbf{F}_q(x_1) < \mathbf{F}_q(x_2). \tag{11}$$

A solution x^{opt} is pareto optimal if it is not dominated by any other solutions. The set of all pareto optimal solutions in the objective space is called pareto optimal front (see [18] for more details). As illustrated in Fig. 2, each point from x_1 to x_8 represents a solution of a bi-objective minimization problem. The points x_1, x_2, x_3, x_4, and x_5 are pareto optimal while points x_6, x_7, and x_8 are not since they are dominated by other points.

4. Our approach to minimize makespan and cost

We design a novel algorithm FDHEFT to solve the multi-objective workflow scheduling problem, which is an extension to the classical list scheduling heuristic HEFT. Below, we first have some discussion on HEFT and HEFT-based algorithms, then introduce the basics of fuzzy dominance sort, and finally present the details about FDHEFT.

4.1. Discussion on HEFT and HEFT-based algorithms

HEFT [14] is the most popular algorithm for obtaining the optimized workflow scheduling on a bounded number of heterogeneous computing resources. As a list-based heuristic, HEFT has two major phases: task prioritizing phase and instance selection phase. In the task prioritizing phase, HEFT calculates the priorities of all tasks and then in the instance selection phase, HEFT selects the tasks in the order of their priorities and schedules each selected task on its best instance to minimize the task's finish time. However, HEFT can only solve the mono-objective workflow scheduling problem. Thus, Durillo et al. [13] extends HEFT to a multi-objective algorithm (MOHEFT) by building several intermediate workflow schedules in parallel in each step, instead of a single one. However, complete coverage traversing is adopted in MOHEFT to generate new solutions for assigning tasks to instances, leading to a large amount of time.

To reduce the time complexity of MOHEFT and obtain the tradeoff solutions with better quality, we design a fuzzy dominance sort based heterogeneous earliest-finish-time algorithm named FDHEFT that incorporates a fuzzy dominance sort based mechanism to determine the best compromised solutions. The proposed FDHEFT is highly effective in exploring tradeoff solutions of high quality and providing fast convergence for our concerned multi-objective problem.

4.2. Basics of fuzzy dominance sort

The concept of pareto optimality is introduced to compare the multiple solutions to an MOP since it is impossible to find a solution that is best with respect to all the objectives. However, all the pareto optimal solutions must be treated as equally good since they are non-dominated. As a result, non-dominance based sorting algorithms would inevitably have the drawback of not providing a complete framework for easy implementation of new methods since the algorithms do not measure the extents by which one solution dominates another [11]. A new measure called fuzzy dominance [19] which correlates more directly with the crisp definition of dominance and has been shown to produce a quick convergence, is proposed to solve the aforementioned defects. In this paper, we design a fuzzy dominance sort based heterogeneous earliest-finish-time algorithm FDHEFT to solve the workflow scheduling problem of minimizing makespan and cost. We introduce the basics of fuzzy dominance sort as follows and show the details of our FDHEFT in Section 4.3.

Suppose the multi-objective minimization problem involves M simultaneous objective functions $f_r(\cdot)$ where $r = 1, 2, \ldots, M$, and let $\Psi \subset \mathbf{R}^\sigma$ be the solution space that contains all the possible solution vectors where σ is the dimensionality. Then, fuzzy dominance related conceptions [19] can be described below.

- **Definition 1: Fuzzy r-dominance by a solution.** Given a monotonically non-decreasing function μ_r^{dom} whose value is in the range of $[0, 1]$, where $r \in \{1, 2, \ldots, M\}$. A solution $u \in \Psi$ is said to r-dominate solution $v \in \Psi$ iff $f_r(u) < f_r(v)$ holds. This relationship can be represented as $u \succ_r^F v$. If $u \succ_r^F v$, the degree of fuzzy r-dominance is equal to

$$\mu_r^{\text{dom}}(f_r(v) - f_r(u)) \equiv \mu_r^{\text{dom}}(u \succ_r^F v). \tag{12}$$

Fuzzy dominance can be regarded as a fuzzy relationship $u \succ_r^F v$ between u and v.

- **Definition 2: Fuzzy dominance by a solution.** Solution $u \in \Psi$ is said to fuzzy dominate solution $v \in \Psi$ iff $\forall r \in \{1, 2, \ldots, M\}$, $u \succ_r^F v$ holds. This relationship can be represented as $u \succ^F v$. The degree of fuzzy dominance can be defined by invoking the concept of fuzzy intersection. If $u \succ^F v$, the degree of fuzzy dominance $\mu^{\text{dom}}(u \succ^F v)$ is obtained by computing the intersection of the fuzzy relationships $u \succ_r^F v$ for each r as

$$\mu^{\text{dom}}(u \succ^F v) = \bigcap_{r=1}^{M} \mu_r^{\text{dom}}(u \succ_r^F v), \tag{13}$$

where $\bigcap$ is the fuzzy intersection operation and $\mu_r^{\text{dom}}(u \succ_r^F v)$ is given in Eq. (12).

- **Definition 3: Fuzzy dominance in a solution set.** Given a solution set $S \subset \Psi$, a solution $v \in S$ is said to be fuzzy dominated in S iff it is fuzzy dominated by any other solution u in S. In such a case, the degree of fuzzy dominance can be calculated by performing a union operation $\bigcup$ over every possible $\mu^{\text{dom}}(u \succ^F v)$, i.e.,

$$\mu^{\text{dom}}(S \succ^F v) = \bigcup_{u \in S} \mu^{\text{dom}}(u \succ^F v), \tag{14}$$

where $\mu^{\text{dom}}(u \succ^F v)$ is given in Eq. (13).

It is clear from Definitions 1–3 that the selection of membership functions $\mu_r^{\text{dom}}(\cdot)$ are crucial to find high quality solutions. According to the implementation of fuzzy dominance [19], the membership functions $\mu_r^{\text{dom}}(\cdot)$ for obtaining the fuzzy r-dominance are required to be zero for negative arguments. In other words, the value of $\mu_r^{\text{dom}}(f_r(v) - f_r(u))$ is necessarily zero if $f_r(u) \geq f_r(v)$. Let Δ_r represent the maximum value of $f_r(v) - f_r(u)$. Similar to the membership functions defined in [11], we set $\mu_r^{\text{dom}}(f_r(v)-f_r(u))$ to 1 if $f_r(v)-f_r(u)=\Delta_r$, indicating that $\mu_r^{\text{dom}}(f_r(v)-f_r(u))$ reaches its maximum value, and set it to the ratio between $f_r(v)-f_r(u)$ and Δ_r. Mathematically, the membership function $\mu_r^{\text{dom}}(f_r(v)-f_r(u))$ is written as

$$\mu_r^{\text{dom}}(\Delta f_r)=\left\{\begin{array}{ll} 0, & \Delta f_r \le 0 \\ \dfrac{\Delta f_r}{\Delta_r}, & 0<\Delta f_r<\Delta_r \\ 1, & \Delta f_r \ge \Delta_r \end{array}\right\}=\mu_r^{\text{dom}}(u \succ_r^F v), \qquad (15)$$

where $\Delta f_r = f_r(v)-f_r(u)$.

After determining the membership functions, each solution can be assigned a single value of fuzzy dominance to reflect the degree that this solution dominates the other solutions in a solution set. The lower the fuzzy dominance value is assigned, the better the solution is. Therefore, sorting solutions according to their fuzzy dominance values can easily help us find the solutions close to the pareto front. In this paper, we use a fuzzy dominance sorting procedure to find and select the best K solutions in each round of solution generation.

4.3. Details on our FDHEFT algorithm

As discussed in Section 4.1, FDHEFT is an extension to the list-based scheduling algorithm HEFT, and thus can be divided into two phases: task prioritizing phase and instance selection phase. In the task prioritizing phase, FDHEFT sorts all the tasks of a given workflow in the non-increasing order of upward rank values, i.e., tasks with larger upward rank values would be assigned higher priorities. In the instance selection phase, FDHEFT sorts all the generated schedule solutions based on their fuzzy dominance values and selects the best K solutions. The pseudo code of FDHEFT is given in Algorithm 1. Before showing the details of Algorithm 1, we first define some key variables used in the algorithm. Let $S=\bigcup_{\ell=1}^{K} S_\ell$ denote the set of K selected tradeoff solutions, where $S_\ell=\bigcup_{i=1}^{n}(T_i, I_s, \text{Type}(I_s))$ is the ℓth selected solution and each of the K selected solutions decides the instance and instance type for n tasks. Since each task can be only executed on one instance, up to n instances are needed for a workflow containing n tasks to ensure that all possible solutions of task-to-instance assignments can be created.

The algorithm takes as inputs a workflow W represented by $G=(T,D)$ and a set of instances represented by $I=\bigcup_{s=1}^{n} I_s$. The n instances can be any type of $R=\bigcup_{k=1}^{m}\{R_k\}$. Similar to HEFT, the algorithm first determines the scheduling order of tasks in set T by $Rank \leftarrow \text{UpRank}(T)$ (line 1), where UpRank() is a function used to calculate the task upward rank values and sort tasks based on upward rank values, and $Rank$ is a scheduling list in which tasks are in the non-increasing order of upward rank value. The details of UpRank function can be found in [14]. The algorithm then initializes the solution set S by $S \leftarrow \{\mathcal{S}'_\ell\}$ and $S'_\ell \leftarrow \emptyset$ (line 2), where S'_ℓ is a temporary solution and is initialized to empty. Afterwards, the algorithm iteratively determines the tradeoff solutions of task-to-instance and instance-to-type assignments for all tasks (lines 3–17). In each round of iteration, the algorithm first employs a temporary set S' to store all possible solutions of assigning task T_i to n instances and determining the types of assigned instances, which is initialized to empty (line 4). The algorithm then creates the possible solutions in the following way. At the beginning the type of each instance is not determined and will be fixed once a task has been assigned to the instance. Thus, when the algorithm creates a solution of assigning task T_i to instance I_s, the algorithm needs to check whether the type of instance I_s is determined. If the type of instance I_s has not been decided, the solution can be generated by binding I_s to an arbitrary type, then S'_ℓ and S' are hence updated (lines 11–14). Otherwise, the solution is directly generated when

Algorithm 1: Fuzzy Dominance Sort Based HEFT

Input: Workflow W, represented by $G=(T,D)$;
A set of n instances with m types, represented by $I=\bigcup_{s=1}^{n} I_s$ and $R=\bigcup_{k=1}^{m}\{R_k\}$;
1 $Rank \leftarrow$ UpRank(T);
2 $S \leftarrow \{\mathcal{S}'_\ell\}$ where $S'_\ell \leftarrow \emptyset$; // $len(S)=1, S_1=\emptyset$
3 **for** $i \leftarrow 1$ *to* n **do**
4   $S' \leftarrow \emptyset$;
5   **for** $\ell \leftarrow 1$ *to* $len(S)$ **do**
6   **for** $s \leftarrow 1$ *to* n **do**
7   **if** type of instance I_s in solution S_ℓ has been decided **then**
8         $S'_\ell \leftarrow S_\ell \cup (Rank_i, I_s, \text{Type}(I_s))$;
9         $S' \leftarrow S' \cup \{\mathcal{S}'_\ell\}$;
10   **else**
11   **for** $k \leftarrow 1$ *to* m **do**
12   Type(I_s) $\leftarrow R_k$;
13           $S'_\ell \leftarrow S_\ell \cup (Rank_i, I_s, \text{Type}(I_s))$;
14           $S' \leftarrow S' \cup \{S'_\ell\}$;
15   **break**;
16   $S' \leftarrow$ FuzzyDomianceSort(S');
17   $S \leftarrow$ SelectFirstKItems(S', K);
18 **return** $S=\bigcup_{\ell=1}^{K} S_\ell$;

task T_i is assigned to instance I_s with a fixed type, then S'_ℓ and S' are also updated (lines 7–9). After obtaining all the possible solutions, the algorithm sorts these solutions based on their fuzzy dominance values by $S' \leftarrow$ FuzzyDomianceSort(S'), and chooses the first K solutions in S' to form S by $S \leftarrow$ SelectFirstKItems(S', K) (lines 16–17). If the assignments of all tasks have been determined, the algorithm exits and returns the set of tradeoff solutions, which is represented by $S=\bigcup_{\ell=1}^{K} S_\ell$ where $S_\ell=\bigcup_{i=1}^{n}(T_i, I_s, \text{Type}(I_s))$ (line 18).

For the purpose of a better understanding of FDHEFT, Fig. 3 gives an example to show the process of generating and selecting solutions for a given task set $T=\{T_1,T_2,T_3\}$ running on an instance set $I=\{I_1,I_2\}$ with type $R=\{R_1,R_2\}$. The three tasks have been sorted using UpRank function and will be assigned in sequence. The number of selected tradeoff solutions used in this example, K, is set to 2. When task T_1 is ready, two possible solutions are constructed and stored in the temporary solution set S', i.e., $S'=\{S_1,S_2\}=\{\{(T_1,I_1,R_1)\},\{(T_1,I_1,R_2)\}\}$. Both solutions are selected and stored in the solution set S using $S' \leftarrow$ FuzzyDomianceSort(S') and $S \leftarrow$ SelectFirstKItems(S', K). In a similar manner, when task T_1 has been assigned and task T_2 is ready, six possible solutions are constructed and stored in set S', and solutions S_2, S_5 are selected from S' and stored in set S. Finally, when tasks T_1 and T_2 have been assigned and task T_3 is ready, eight solutions are constructed and stored in set S', and solutions S_1, S_4 are selected from S' and stored in set S. As a result, $S=\{S_1,S_4\}=\{\{(T_1,I_1,R_1),(T_2,I_2,R_1),(T_3,I_1,R_1)\},\{(T_1,I_1,R_2),(T_2,I_2,R_1),(T_3,I_3,R_2)\}\}$ is the set of tradeoff solutions obtained by FDHEFT. Note that we focus on presenting the high-level overview of FDHEFT and thus omit introducing how to perform fuzzy dominance sort in this example.

The fuzzy dominance sort FuzzyDomianceSort() consists of two major parts, i.e., fuzzy dominance value calculation (FDVC) procedure used to derive the fuzzy dominance values of solutions and perimeter value assignment (PVA) procedure used to handle the solutions with identical dominance values. The details about these two procedures are described in Algorithms 2 and 3, respectively.

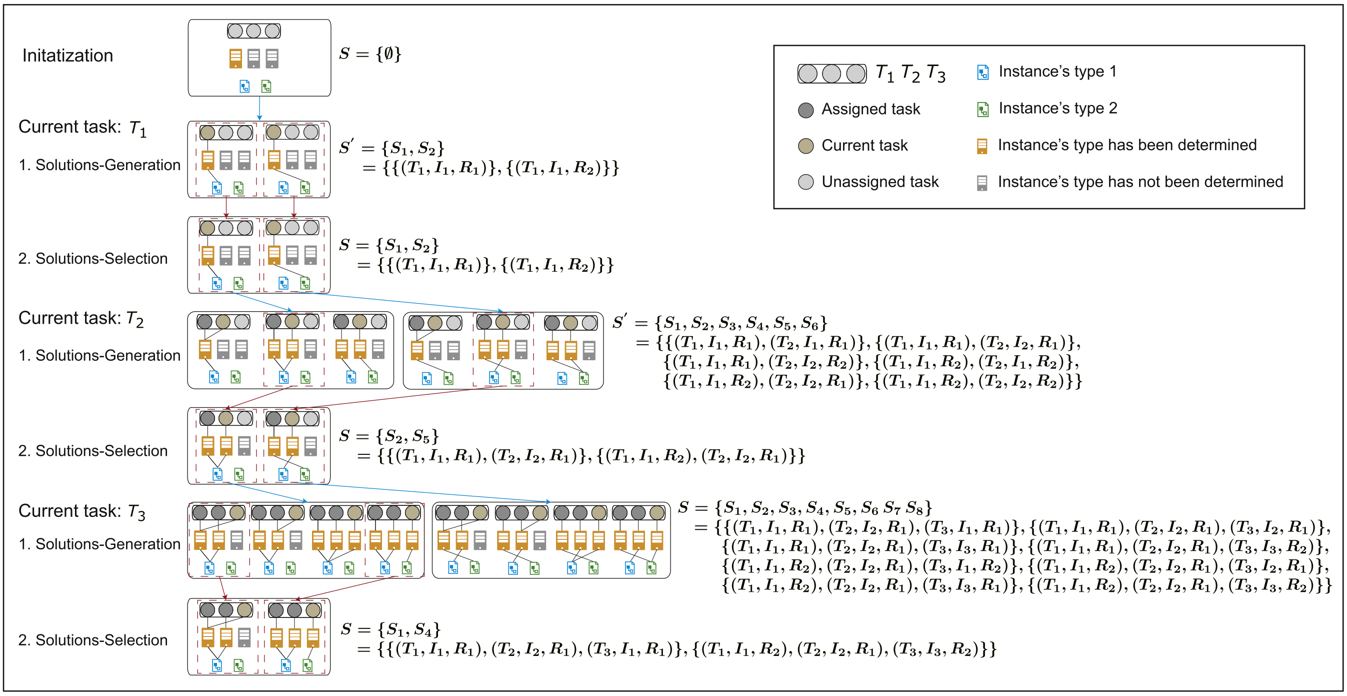


Fig. 3. An example to show the high-level overview of FDHEFT.

Algorithm 2: Fuzzy Dominance Value Calculation

Input: Solution set S;
1 **for** $\ell \leftarrow 1$ *to* $len(S)$ **do**
2   $\kappa \leftarrow 0$; // a temporary variable to store the value of $\mu^{\text{dom}}(S \succ^F S_\ell)$
3   **for** $j \leftarrow 1$ *to* $len(S)$ **do**
4   **if** $\ell = j$ **then**
5   **continue**;
6     $\mu \leftarrow 1$;
7   **if** $S_j \succ S_\ell$ **then**
8   **for** $r \leftarrow 1$ *to* M **do**
9   derive $\mu_\ell^{\text{dom}}(f_r(S_\ell) - f_r(S_j))$ using Eqs. (12) and (15);
10   according to Eq. (13), calculate μ by $\mu \leftarrow \mu \cap \mu_\ell^{\text{dom}}(f_r(S_\ell) - f_r(S_j))$;
11   according to Eq. (14), obtain κ by $\kappa \leftarrow \kappa \cup \mu$;
12   $\mu^{\text{dom}}(S \succ^F S_\ell) \leftarrow \kappa$;

Algorithms 2 iteratively determines the fuzzy dominance value of each solution in a given solution set S based on Definitions 1–3 presented in Section 4.2. The algorithm mainly operates as follows. It uses a temporary variable κ to store the value of fuzzy dominance in a solution set represented by $\mu^{\text{dom}}(S \succ^F S_\ell)$ (lines 2 and 12). To derive the fuzzy dominance in a solution set using Eq. (14), it needs to calculate the fuzzy r-dominance by a solution using Eqs. (12) and (15) and to compute the fuzzy dominance by a solution using Eq. (13) at first (lines 9–10).

After calculating the fuzzy dominance values of solutions, one immediate question is how to determine which solution is better if some solutions have the same fuzzy dominance value. To handle this case, we employ the diversity fitness function [20] as the criteria to compare the solutions with the same fuzzy dominance value, which equals to the perimeter of the largest M dimensional hypercube in the objective space. The perimeter of a solution indicates the region of sparsity along the solution. Given a solution S_ℓ in set S, the perimeter value of S_ℓ is

$$P(S_\ell) = \sum_{r=1}^{M} \frac{f_r(u) - f_r(v)}{\max f_r - \min f_r} \tag{16}$$

where M is the number of objective functions, and u and v are solutions that are adjacent to solution S_ℓ with the same fuzzy dominance value. $f_r(u)$ and $f_r(v)$ are the values of rth objective of solutions u and v, respectively. $\max f_r$ and $\min f_r$ are the maximal and minimal values of rth objective of all solutions in set S.

Algorithm 3: Perimeter Value Assignment

Input: Solution set S;
1 $L \leftarrow len(S)$;
2 **for** $\ell \leftarrow 1$ *to* L **do**
3   $P(S_\ell) \leftarrow 0$;
4 **for** $r \leftarrow 1$ *to* M **do**
5   $S \leftarrow \text{Sort}(S, f_r)$; // sort all solutions in set S based on their values of objective f_r
6   $P(S_1) \leftarrow \infty$;
7   $P(S_L) \leftarrow \infty$;
8   **for** $\ell \leftarrow 2$ *to* $(L-1)$ **do**
9     $P(S_\ell) \leftarrow P(S_\ell) + \frac{S^r_{\ell+1} - S^r_{\ell-1}}{\max S^r - \min S^r}$; // derive $P(S_\ell)$ using Eq. (16)

According to the definition of diversity fitness function [20], the perimeters of boundary solutions on the tradeoff optimal front are assigned infinity. In addition, solutions with higher perimeter values are preferred since using this way is beneficial to maintaining the diversity of the solutions. The pseudo code of PVA is given in Algorithm 3. It first calculates the number L of solutions in set S and initializes the perimeter values of all solutions to zero (lines 1–3), and then iteratively derives the perimeter values of all solutions with respect to their M objectives (line 4–9). In each round of iteration, the algorithm sorts all solutions in set S based on their values of objective f_r (line 5), sets the perimeters of

Table 1
Characteristics of the real-world workflows.

Workflow	Number of nodes	Number of edges	Average data size	Average task runtime (CU=1)
CyberShake 30	30	112	747.48 MB	23.77 s
CyberShake 50	50	188	864.74 MB	29.32 s
CyberShake 100	100	380	849.60 MB	31.53 s
Inspiral 30	30	95	9.00 MB	206.78 s
Inspiral 50	50	160	9.16 MB	226.19 s
Inspiral 100	100	319	8.93 MB	206.12 s
Montage 25	25	95	3.43 MB	8.44 s
Montage 50	50	206	3.36 MB	9.78 s
Montage 100	100	433	3.23 MB	10.58 s
Sipht 30	30	91	7.73 MB	178.92 s
Sipht 60	60	198	6.95 MB	194.48 s
Sipht 100	100	335	6.27 MB	175.55 s

boundary solutions S_1 and S_L to infinity (lines 6–7), and calculates the perimeters of remainder solutions using Eq. (16) (lines 8–9).

5. Evaluation

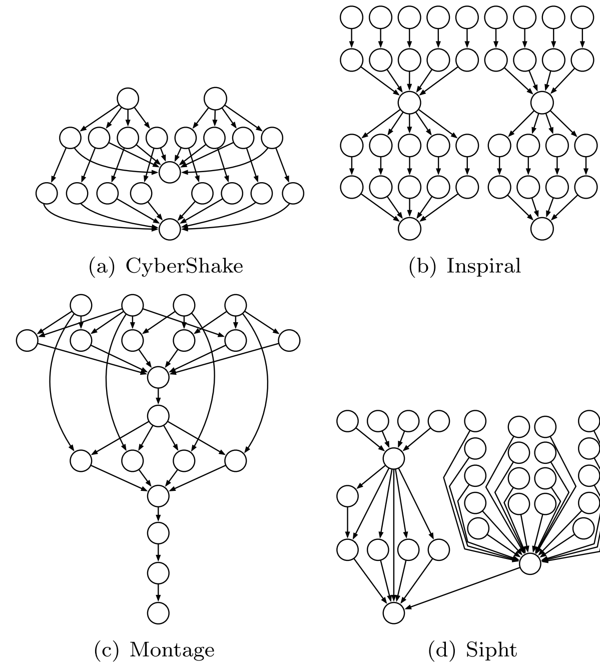
We validate our cost- and makespan-aware workflow scheduling scheme through two sets of simulation experiments. The first set of simulations is based on real-world workflows while the second set of simulations is based on synthetic applications. This section introduces the experimental setups and analyzes the experimental results.

5.1. Simulation setup

Below, we describe the setups of real-world and synthetic workflows, IaaS model, comparative algorithms, simulator, and performance metrics used in the simulations.

5.1.1. Workflows

In the first set of simulations, we consider four types of real-world benchmark workflows provided by the Pegasus workflow management system [21,22], i.e., CyberShake, Inspiral, Montage, and Sipht. These workflows are widely used for evaluating the performance of scheduling algorithms. The DAG characteristics of these workflows, including the number of nodes and edges, average data size and task execution time at CU = 1, are presented in Table 1. The structures of these four workflow applications are illustrated in Fig. 4. In the second set of simulations, we test our scheme and peer algorithms on synthetic DAGs. Same as in [4,23], we utilize a widely-used Workflow Generator [24] to produce 100 synthetic workflows, which follows the real-world workflow types provided by the Pegasus workflow management system [22] and varies the number of workflow nodes ranging from 20 to 100. To obtain realistically synthetic workflows, the generator uses the information gathered from actual executions of scientific workflows on the Grid. In addition, the generator is highly encapsulated and its inputs are workflow type and node number. Once the type and node number of a desired synthetic workflow are set, the rest characteristics of this workflow such as the edge number and the average data size as well as the computation and communication costs of tasks in this workflow, are automatically determined. The determination process (including how to calculate task computation and communication costs) is actually similar to the calculation method introduced in this paper and thus is not discussed in the manuscript. For more details refer to the source-code provided in [25].



(a) CyberShake (b) Inspiral (c) Montage (d) Sipht

Fig. 4. Structures of four real-world workflows.

Table 2
IaaS parameters used in the experiments.

Instance type	Compute unit	Bandwidth (Mb/s)	Price ($)
m4.large	6.5	56.25	0.120
m4.xlarge	13	93.75	0.239
m4.2xlarge	26	125	0.479
m4.4xlarge	53.5	250	0.958
m4.10xlarge	124.5	500	2.394
m3.medium	3	56.25	0.067
m3.large	6.5	56.25	0.133
m3.xlarge	13	62.5	0.266
m3.2xlarge	26	125	0.532

5.1.2. IaaS model

We select the pricing scheme of Amazon EC2 as our pricing model due to its widespread application. Among the various types of instances provided by Amazon EC2, the General Purpose instance group in US East region with the purchasing option of On-Demand Instance [26] is used. The relevant parameters including instance type, compute unit, bandwidth, and price are listed in Table 2. Note that in the experiments we only employ the instances provided by US East region since using Amazon Web Services (AWS) to run Pegasus workflows requires us to stick to one Region [27] in order to avoid failures occurring across regions. As a result, the monetary cost of data transfers between instances in different regions are not included.

5.1.3. Comparative algorithms

We compare our FDHEFT with several multi-objective optimization algorithms, including ε-Fuzzy PSO [11], MOHEFT [13], NSPSO [10], and SPEA2* [12]. These comparative algorithms are described below.

- ε-Fuzzy PSO [11] is an improved version of PSO. It also utilizes fuzzy dominance to measure the relative fitness of solutions in multi-objective domain, which has been proven to be highly effective and can provide faster convergence for most difficult MOPs.
- NSPSO [10] is also a variant of PSO. Unlike PSO, NSPSO can discover more non-dominated relations by comparing the

Table 3
HV of five algorithms (our FDHEFT and peer algorithms ε-Fuzzy PSO [11], NSPSO [10], SPEA2* [12], and MOHEFT [13]) on real-world workflows, and HV differences between FDHEFT and ε-Fuzzy PSO, NSPSO, SPEA2*, and MOHEFT.

Workflow	HV of five algorithms					HV difference (%, against FDHEFT)			
	ε-Fuzzy PSO	NSPSO	SPEA2*	MOHEFT	FDHEFT	ε-Fuzzy PSO	NSPSO	SPEA2*	MOHEFT
CyberShake 30	0.615	0.954	0.905	0.945	**0.961**	56.1	0.7	6.1	1.6
CyberShake 50	0.917	0.932	0.876	0.929	**0.954**	4.0	2.4	8.9	2.7
CyberShake 100	0.822	0.863	0.871	0.935	**0.949**	15.4	9.9	8.9	1.5
Inspiral 30	0.716	0.365	0.762	0.787	**0.847**	18.2	132.2	11.1	7.6
Inspiral 50	0.408	0.652	0.634	0.785	**0.846**	107.5	29.6	33.4	7.7
Inspiral 100	0.269	0.200	0.502	0.697	**0.782**	191.2	291.2	55.9	12.3
Montage 25	0.944	0.945	0.915	0.947	**0.960**	1.6	1.6	4.9	1.3
Montage 50	0.721	0.637	0.916	0.966	**0.978**	35.6	53.4	6.8	1.3
Montage 100	0.429	0.928	0.813	0.934	**0.961**	124.0	3.6	18.2	2.9
Sipht 30	0.126	0.398	0.117	0.519	**0.519**	311.0	30.3	342.5	0.0
Sipht 60	0.200	0.595	0.391	0.722	**0.762**	281.2	28.1	94.8	5.6
Sipht 100	0.279	0.200	0.347	0.820	**0.837**	200.0	318.3	141.3	2.1

Table 4
Runtime of five algorithms (our FDHEFT and peer algorithms ε-Fuzzy PSO [11], NSPSO [10], SPEA2* [12], and MOHEFT [13]) on real-world workflows, and runtime ratios of ε-Fuzzy PSO, NSPSO, SPEA2*, and MOHEFT against FDHEFT.

Workflow	Runtime of five algorithms (s)					Runtime ratios (against FDHEFT)			
	ε-Fuzzy PSO	NSPSO	SPEA2*	MOHEFT	FDHEFT	ε-Fuzzy PSO	NSPSO	SPEA2*	MOHEFT
CyberShake 30	13.96	13.13	5.12	9.55	**0.13**	104.14	98.01	38.21	71.28
CyberShake 50	36.99	35.31	5.69	43.51	**0.34**	107.53	102.65	16.54	126.47
CyberShake 100	143.12	138.12	7.46	555.31	**1.51**	94.85	91.53	4.95	368.00
Inspiral 30	13.06	12.38	5.93	11.93	**0.54**	24.37	23.09	11.06	22.25
Inspiral 50	36.90	34.20	5.50	50.56	**0.50**	74.24	68.81	11.06	101.72
Inspiral 100	139.05	134.09	7.02	622.71	**1.19**	116.56	112.40	5.89	521.97
Montage 25	10.00	8.93	5.15	5.60	**0.11**	92.60	82.64	47.67	51.88
Montage 50	36.05	34.21	5.32	46.30	**0.33**	107.94	102.43	15.94	138.63
Montage 100	140.00	135.37	6.52	552.32	**1.74**	80.32	77.66	3.74	316.88
Sipht 30	12.74	11.94	7.97	10.52	**0.12**	104.43	97.89	65.30	86.26
Sipht 60	48.57	45.66	8.62	74.72	**0.40**	122.97	115.61	21.83	189.18
Sipht 100	133.23	130.15	7.16	518.55	**0.94**	142.49	139.20	7.66	554.60

personal bests and offspring of all particles in a combined swarm population, providing a more appropriate selection pressure for the population to approach the true pareto-optimal front.

- MOHEFT [13] is an improved version of HEFT. It employs dominance relationships and a metric called crowding distance to avoid the exhaustive search and the computationally expensive approach for pruning the set of tradeoff solutions.
- SPEA2* [12] is an improved version of SPEA2. It redefines the crossover and mutation operators of SPEA2 such that the resultant evolutionary algorithm can search better solutions more efficiently.

5.1.4. Simulator

We performed all the experiments on a desktop PC equipped with a 3.3 GHz Intel Core i5 CPU and 16 GB RAM. Considering that our FDHEFT and the comparative algorithms are all MOP methods, we adopt a common simulator jMetal [28], an object-oriented Java-based framework aimed at the development, experimentation, and study of metaheuristics for solving MOP problems, to implement all the algorithms. jMetal is featured by including a number of classic and modern state-of-the-art optimizers, a wide set of benchmark problems, and a set of well-known quality indicators for performance assessment. The inputs to jMetal are the algorithm to be implemented and its associated setup. We list the key parameters of the algorithms implemented in jMetal below.

- For MOHEFT [13] and our FDHEFT, the number of trade-off solutions are 30 and 50, respectively.
- For all ε-Fuzzy PSO [11], MOHEFT [13], NSPSO [10], and SPEA2* [12], the size of population is 100 and the number of generations is 10 000.
- For SPEA2* [12], the probabilities of crossover and mutation are 0.9 and 0.2, respectively.
- For both ε-Fuzzy PSO and NSPSO, we use the same parameters as in [11] and [10], respectively.

5.1.5. Performance metric

Three comparative experiments are carried out in each set of simulations to validate our scheme from different perspectives. First, we compare the Hypervolume (HV) [29] of our FDHEFT with that of four comparative algorithms. HV [29] is one of the most widely-used performance metrics in the multi-objective optimization area. It is calculated as the volume of the objective space between the obtained solution set and the reference point and thus can provide a combined information about convergence and diversity of the solution set. According to its calculation, a larger HV value is preferred, indicating that the solution set is closer to the pareto front and also has a good distribution. Second, we compare the CPU runtime taken by FDHEFT with that of four comparative algorithms. Third, we compare the cost-makespan trade-off fronts obtained by FDHEFT with that of four comparative algorithms.

5.2. Results and analysis

We discuss the results of our simulations verified on both real-world and synthetic workflows.

5.2.1. Results of real-world workflows

Table 3 presents the HV results achieved by our FDHEFT and four comparative algorithms ε-Fuzzy PSO [11], NSPSO [10], SPEA2* [12], and MOHEFT [13] on 12 real-world workflows. For a more intuitive comparison, the table also shows the HV differences between FDHEFT and ε-Fuzzy PSO, NSPSO, SPEA2*, and MOHEFT, which are readily derived from the HV results. For example, the

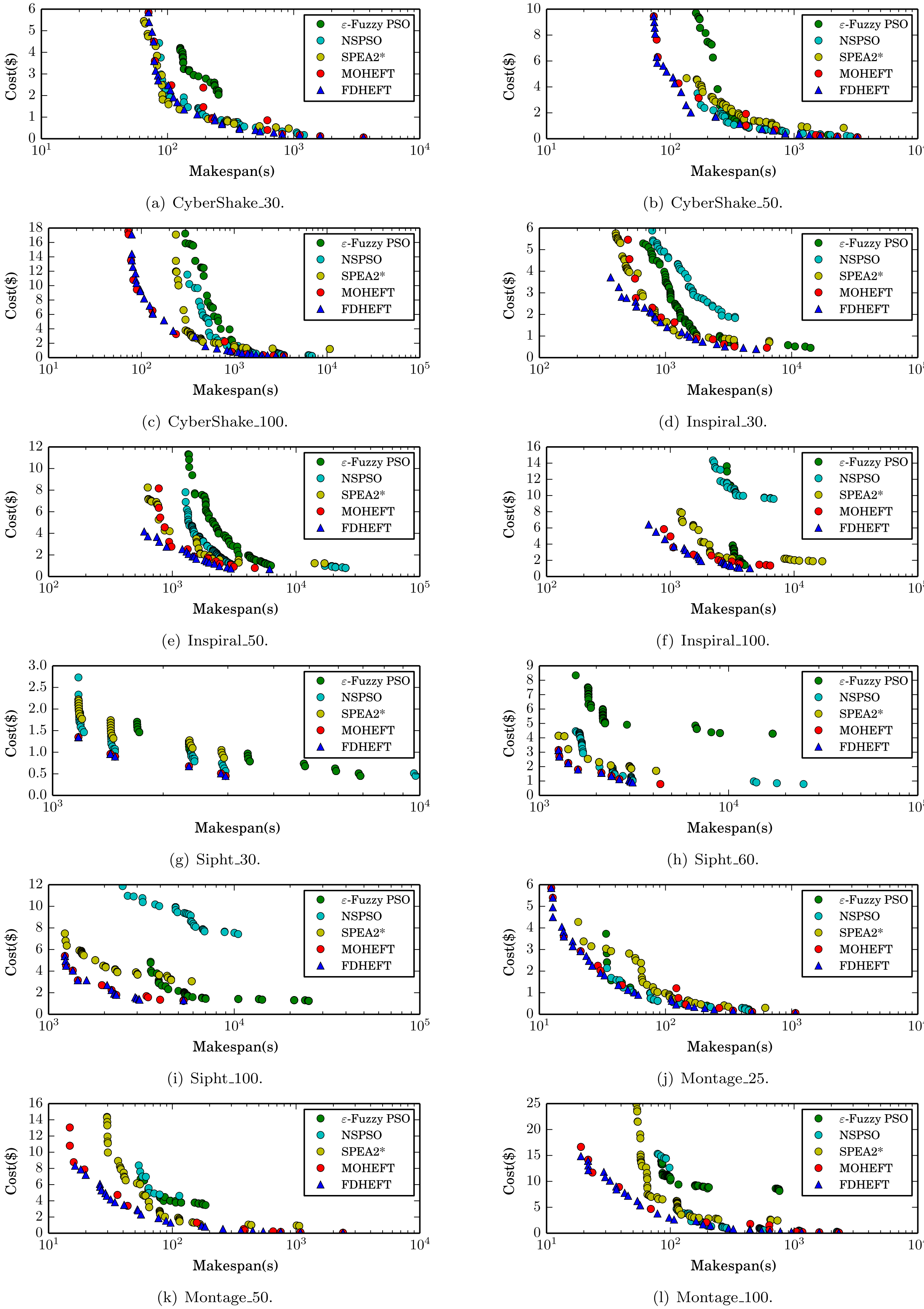


Fig. 5. Cost-makespan trade-offs for our FDHEFT and four peer algorithms on real-world workflows.

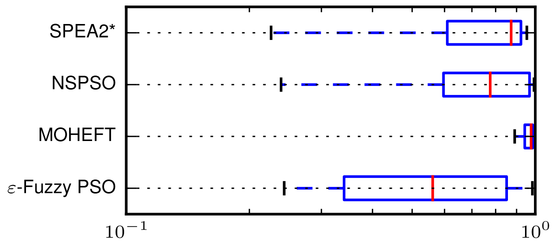


Fig. 6. Box plots for the HV ratios of SPEA2* [12], NSPSO [10], MOHEFT [13], and ε-Fuzzy PSO [11] against FDHEFT on 100 random workflows.

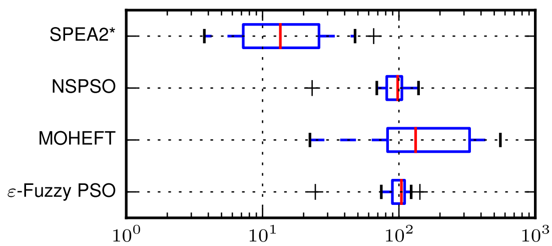


Fig. 7. Box plots for the runtime ratios of SPEA2* [12], NSPSO [10], MOHEFT [13], and ε-Fuzzy PSO [11] against FDHEFT on 100 random workflows.

HV difference between FDHEFT and ε-Fuzzy PSO, represented by Diff(ε-Fuzzy PSO), is calculated as

$$\text{Diff}(\varepsilon\text{-Fuzzy PSO}) = \frac{\text{HV(FDHEFT)}}{\text{HV}(\varepsilon\text{-Fuzzy PSO})} \times 100\% - 1.$$

Apparently, the HV difference is positive when our FDHEFT outperforms a comparative algorithm and negative otherwise. Thus, HV difference can be considered as the improvement achieved by our FDHEFT. As can be seen from the table, all HV differences are positive, indicating that our FDHEFT always have a better performance than ε-Fuzzy PSO, NSPSO, SPEA2*, and MOHEFT regardless of which workflows used in the experiment. The HV improvements achieved by our FDHEFT over ε-Fuzzy PSO, NSPSO, SPEA2*, and MOHEFT can be up to 311%, 318.3%, 342.5%, and 12.3%, respectively.

Table 4 gives the CPU runtime statistics by our FDHEFT and comparative algorithms ε-Fuzzy PSO [11], NSPSO [10], SPEA2* [12], and MOHEFT [13] for executing 12 real-world workflows. The results show that our FDHEFT is much more computationally efficient than the four comparative algorithms. Specifically, the CPU runtime of our FDHEFT and four comparative algorithms are in the ranges of [12.74 s, 143.12 s], [8.93 s, 138.12 s], [5.12 s, 8.62 s], [5.60 s, 622.71 s], and [0.11 s, 1.74 s], respectively. The table also lists the runtime ratios of the four comparative algorithms against our FDHEFT, which directly validate the time efficiency of our algorithms. For example, the CPU runtime of MOHEFT can be dozens or even hundreds of times higher than that of our FDHEFT. From the table, we can also find that the execution time of all algorithms increases with the number of tasks in the application.

Fig. 5 plots the produced cost-makespan trade-offs for different algorithms, our FDHEFT and four comparative algorithms ε-Fuzzy PSO [11], NSPSO [10], SPEA2* [12], and MOHEFT [13], on real-world workflows CyberShake, Inspiral, Sipht, and Montage with varying number of nodes. Note that all the x-axes are logarithmic and each point on the plot represents a possible task schedule.

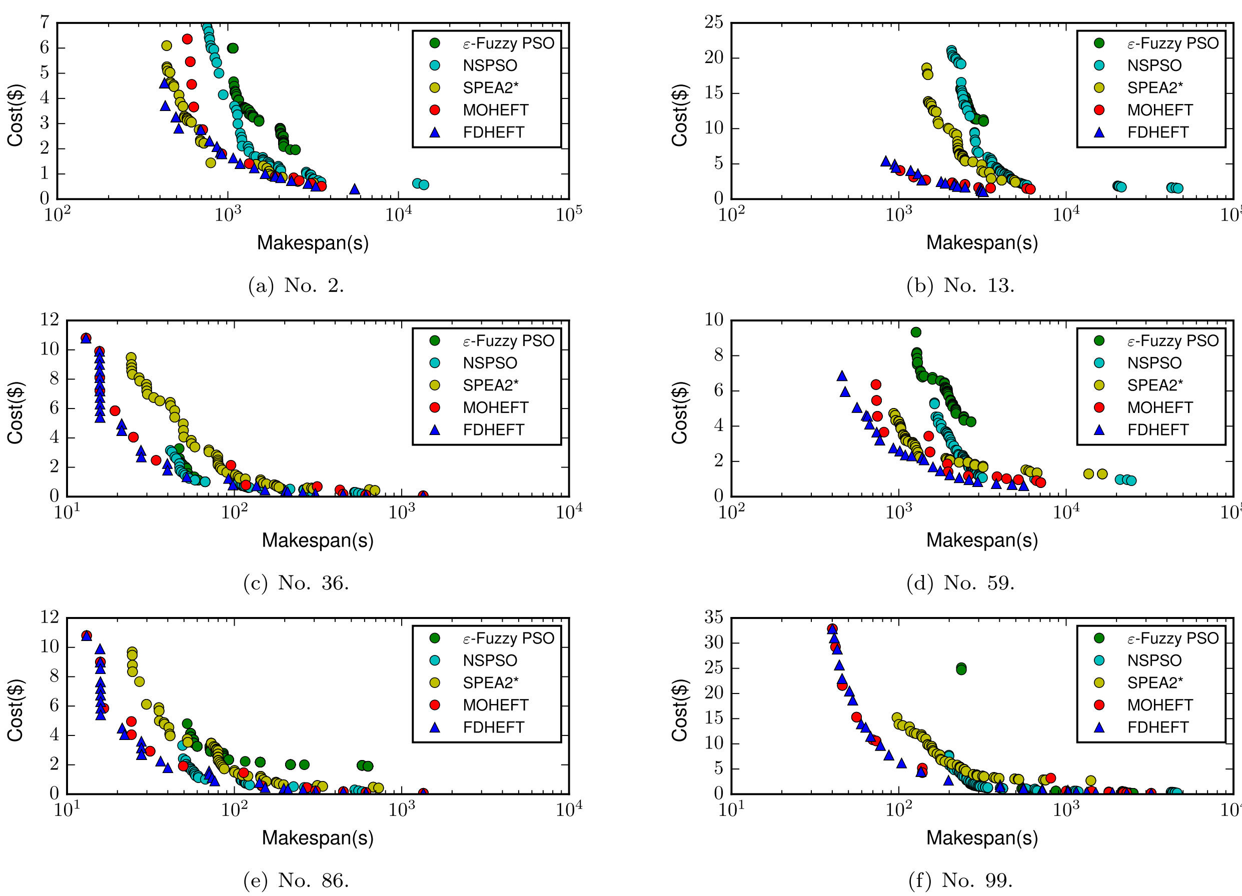


Fig. 8. Cost-makespan trade-offs for our FDHEFT and four peer algorithms on selected synthetic workflows.

The plots given in Fig. 5 indicate that the trade-off fronts obtained by our FDHEFT are significantly superior to those obtained by the comparative algorithms. In other words, the task schedules generated by our FDHEFT can provide better cost-makespan trade-offs.

5.2.2. *Results of synthetic workflows*

We also evaluate our FDHEFT and four peer algorithms on 100 synthetic applications in the following three steps. First, we compare the HV ratios of peer algorithms against our FDHEFT on each tested synthetic application. If the ratio is less than 1, our FDHEFT is shown to perform better than the competitor, and worse otherwise. Then, we compare the cost-makespan trade-off plots achieved by our FDHEFT and four peer algorithms for a set of randomly selected synthetic applications. Finally, we compare the runtime ratios of our FDHEFT and peer algorithms on each tested synthetic application. If the ratio is larger than 1, our FDHEFT is shown to be more time-efficient than the competitor, and less otherwise.

Fig. 6 shows the box plots for the HV ratios of ε-Fuzzy PSO [11], NSPSO [10], SPEA2* [12], and MOHEFT [13] against our FDHEFT on 100 synthetic applications. The results in the figure clearly show that our FDHEFT is slightly better than SPEA2* and MOHEFT, and remarkably better than ε-Fuzzy PSO and NSPSO, respectively. Fig. 7 presents the box plots for the runtime ratios of ε-Fuzzy PSO, NSPSO, SPEA2*, and MOHEFT against our FDHEFT on 100 synthetic applications. As can be seen in the figure, the runtime ratios of four peer algorithms are all above 1, indicating that our FDHEFT is the fastest algorithm. The time overhead of our FDHEFT for generating solutions is only a small percentage of that of the peer algorithms. Fig. 8 gives the cost-makespan trade-off plots of our FDHEFT and four peer algorithms for some randomly selected synthetic applications (Nos. 2, 13, 36, 59, 86, 99). The plots in the figures show that our FDHEFT can obtain trade-off fronts with clear advantage over the competitors.

6. Conclusion

In this paper, we aimed to minimize cost and makespan simultaneously for workflows deployed and hosted on IaaS clouds. For a workflow with precedence constraints among tasks, we proposed a new list scheduling algorithm FDHEFT that integrates the fuzzy dominance sort mechanism with heuristic HEFT. Two sets of simulation experiments were implemented on real-world workflows and synthetic applications to validate the effectiveness of the proposed FDHEFT. The extensive experiments are based on the actual pricing and resource parameters of Amazon EC2, and results have demonstrated that the proposed FDHEFT can explore better makespan-cost trade-offs for scheduling workflows with a much lower CPU runtime when compared to a number of peer approaches. In future, we plan to extend our algorithm to solve the workflow scheduling problem when considering the monetary costs and time overheads of both communication and storage.

References

[1] P. Cong, L. Li, J. Zhou, K. Cao, T. Wei, M. Chen, S. Hu, Profit-driven dynamic cloud pricing for multiserver systems considering user perceived value, IEEE Trans. Parallel Distrib. Syst. (2018).

[2] T. Wu, H. Gu, J. Zhou, T. Wei, X. Liu, M. Chen, Soft error-aware energy-efficient task scheduling for workflow applications in DVFS-enabled cloud, J. Syst. Archit. 84 (2018) 12–27.

[3] X. Zhang, T. Wu, M. Chen, T. Wei, J. Zhou, S. Hu, R. Buyya, Energy-aware virtual machine allocation for cloud with resource reservation, J. Syst. Softw. 147 (2019) 147–161.

[4] Z. Zhu, G. Zhang, M. Li, X. Liu, Evolutionary multi-objective workflow scheduling in cloud, IEEE Trans. Parallel Distrib. Syst. 27 (5) (2016) 1344–1357.

[5] Amazon Case Studies. [Online] Available: https://aws.amazon.com/cn/solutions/case-studies/.

[6] Windows Azure Case Studies. [Online] Available: https://www.microsoft.com/azure/casestudies.mspx.

[7] I. Casas, J. Taheri, R. Ranjan, L. Wang, A.Y. Zomaya, A balanced scheduler with data reuse and replication for scientific workflows in cloud computing systems, Future Gener. Comput. Syst. 74 (2017) 168–178.

[8] E.N. Alkhanak, S.P. Lee, S.U.R. Khan, Cost-aware challenges for workflow scheduling approaches in cloud computing environments: taxonomy and opportunities, Future Gener. Comput. Syst. 50 (2015) 3–21.

[9] S. Su, J. Li, Q. Huang, X. Huang, K. Shuang, J. Wang, Cost-efficient task scheduling for executing large program in the cloud, Parallel Comput. 39 (4) (2013) 177–188.

[10] R. Garg, A.K. Singh, Multi-objective workflow grid scheduling based on discrete particle swarm optimization, in: Swarm, Evolutionary, and Memetic Computing, 2011, pp. 183–190.

[11] R. Garg, A.K. Singh, Multi-objective workflow grid scheduling using ε-fuzzy dominance sort based discrete particle swarm optimization, J. Supercomput. 68 (2) (2014) 709–732.

[12] J. Yu, M. Kirley, R. Buyya, Multi-objective planning for workflow execution on grids, in: IEEE/ACM International Conference on Grid Computing, 2007, pp. 10–17.

[13] J.J. Durillo, R. Prodan, Multi-objective workflow scheduling in Amazon EC2, Clust. Comput. 17 (2) (2014) 169–189.

[14] H. Topcuoglu, S. Hariri, M.Y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Trans. Parallel Distrib. Syst. 13 (3) (2002) 260–274.

[15] G. Xie, G. Zeng, J. Jiang, C. Fan, R. Li, K. Li, Energy management for multiple real-time workflows on cyber-physical cloud systems, Future Gener. Comput. Syst. (2018).

[16] X. Zhu, J. Wang, H. Guo, D. Zhu, L.T. Yang, L. Liu, Fault-tolerant scheduling for real-time scientific workflows with elastic resource provisioning in virtualized clouds, IEEE Trans. Parallel Distrib. Syst. 27 (12) (2016) 3501–3517.

[17] G. Xie, G. Zeng, R. Li, K. Li, Quantitative fault-tolerance for reliable workflows on heterogeneous IaaS clouds, IEEE Trans. Cloud Comput. (2018).

[18] K. Deb, Multi-objective optimiation using evoluationary algorithms, in: Wiley-Interscience Series in Systems and Optimization, Wiley, Chichester, 2001.

[19] P. Koduru, Z. Dong, S. Das, S.M. Welch, J.L. Roe, E. Charbit, A multiobjective evolutionary-simplex hybrid approach for the optimization of differential equation models of gene networks, IEEE Trans. Evol. Comput. 12 (5) (2008) 572–590.

[20] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: Nsga-ii, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.

[21] E. Deelman, et al., Pegasus: A framework for mapping complex scientific onto distributed systems, Sci. Program. 13 (3) (2005) 219–237.

[22] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, K. Vahi, Characterizing and profiling scientific workflows, Future Gener. Comput. Syst. 29 (3) (2013) 682–692.

[23] H. Arabnejad, J.G. Barboss, A budget constrained scheduling algorithm for workflow applications, J. Grid Comput. 12 (2014) 665–679.

[24] Workflow Generator, [Online] Available: https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator.

[25] Source-Code of Workflow Generator, [Online] Available: https://github.com/pegasus-isi/WorkflowGenerator.

[26] Amazon, Amazon ec2 pricing, [Online] Available: http://aws.amazon.com/ec2/pricing/.

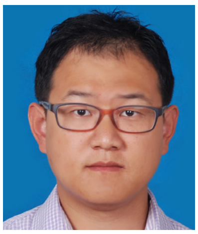
[27] Pegasus, [Online] Available: https://confluence.pegasus.isi.edu/display/pegasus/Cloud+Tutorial.

[28] J.J. Durillo, A.J. Nebro, jmetal: A java framework for multi-objective optimization, Adv. Eng. Softw. 42 (2011) 760–771.

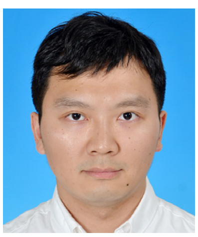
[29] E. Zitzler, L. Thiele, Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach, IEEE Trans. Evol. Comput. 3 (4) (1999) 257–271.

Xiumin Zhou received the B.S. degree in computer science and Engineering from Nanjing University of Science and Technology, Nanjing, China, in 2012. He is currently pursuing the Ph.D. degree with Nanjing University of Science and Technology. His current research interests are in the areas of embedded systems, cloud computing and distributed systems.

Gongxuan Zhang received the B.S. degree in electronic computer from Tianjin University in 1983 and the M.S. and Ph.D. degrees in Computer Application from the Nanjing University of Science and Technology in 1991 and 2005, respectively. He was a Senior Visiting Scholar in Royal Melbourne Institute of Technology from 2001.9 to 2002.3 and University of Notre Dame from 2017.7 to 2017.10. Since 1991, he has been with the Nanjing University of Science and Technology, where he is currently a professor in the School of Computer Science and Engineering. His current research interests include multicore and parallel processing and distributed computing. He is a senior member of IEEE.



Jin Sun received the B.S. and M.S. degrees in computer science from Nanjing University of Science and Technology, Nanjing, China, in 2004 and 2006, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Arizona in 2011. Between January 2012 and September 2014, he was with Orora Design Technologies, Inc. as a member of technical staff. He is currently an Associate Professor with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. His research interests include integrated circuit modeling and analysis and computer-aided design. Dr. Sun has been an Associate Editor for the Journal of Circuits, Systems, and Computers since 2018. He is a member of IEEE.



Junlong Zhou received the Ph.D. degree in Computer Science from East China Normal University, Shanghai, China, in 2017. He was a Visiting Scholar with the University of Notre Dame, Notre Dame, IN, USA, during 2014-2015. He is currently an Assistant Professor with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. His research interests include real-time embedded systems, cloud computing and IoT, and cyber–physical systems. Dr. Zhou is an Active Reviewer of 20 international journals, including IEEE Transactions on Computers, IEEE Transactions on CAD of Integrated Circuits and Systems, IEEE Transactions on Industrial Informatics, IEEE Transactions on Systems, Man and Cybernetics: Systems, ACM Transactions on Design Automation of Electronic Systems, and ACM Transactions on Cyber–Physical Systems. He received the Reviewer Award from Journal of Circuits, Systems, and Computers, in 2016. Dr. Zhou has been an Associate Editor for the Journal of Circuits, Systems, and Computers since 2017, and a Guest Editor for the ACM Transactions on Cyber–Physical Systems in 2018. He is a member of IEEE.

Tongquan Wei received the Ph.D. degree in electrical engineering from Michigan Technological University, Houghton, MI, USA, in 2009. He is currently an Associate Professor with the Department of Computer Science and Technology, East China Normal University, Shanghai, China. His current research interests include real-time embedded systems, green and reliable computing, parallel and distributed systems, and cloud computing. Dr. Wei has been a Regional Editor for the Journal of Circuits, Systems, and Computers since 2012. He served as a Guest Editor for several special sections of the IEEE Transactions on Industrial Informatics, ACM Transactions on Embedded Computing Systems, and ACM Transactions on Cyber–Physical Systems. He is a member of IEEE.

Shiyan Hu received his Ph.D. in Computer Engineering from Texas A&M University in 2008. He is an Associate Professor at Michigan Technological University. He has been a Visiting Professor at IBM Research (Austin) in 2010, and a Visiting Associate Professor at Stanford University from 2015 to 2016. His research interests include Cyber–Physical Systems (CPS), CPS Security, Computer Aided Design of VLSI Circuits, and Embedded Systems, where he has published more than 100 refereed papers. He is an ACM Distinguished Speaker, an IEEE Systems Council Distinguished Lecturer, an invited participant for U.S. National Academy of Engineering Frontiers of Engineering Symposium, a recipient of National Science Foundation (NSF) CAREER Award, a recipient of ACM SIGDA Richard Newton DAC Scholarship (as the faculty advisor), and a recipient of JSPS Faculty Invitation Fellowship. He is the Chair for IEEE Technical Committee on Cyber–Physical Systems. He is the Editor-In-Chief of IET Cyber–Physical Systems: Theory & Applications. He serves as an Associate Editor for IEEE Transactions on Computer-Aided Design, IEEE Transactions on Industrial Informatics, and IEEE Transactions on Circuits and Systems. He is also a Guest Editor for 7 IEEE/ACM Transactions such as Proceedings of IEEE and IEEE Transactions on Computers. He has served as general chairs, TPC chairs, TPC track chairs and TPC members for numerous conferences. He is a Fellow of IET.