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

Xiumin Zhou, Gongxuan Zhang, Jin Sun, Junlong Zhou, Tongquan Wei, Shiyan Hu

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 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...
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 \(\varepsilon\)-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 \(\varepsilon\)-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 the 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 conduct extensive simulation experiments to validate FDHEFT. We compare our FDHEFT with a number of representative multi-objective optimization algorithms, including NSPSO [10], \(\varepsilon\)-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.

- We formulate the problem of jointly minimizing monetary cost and makespan 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.

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 \(Pre(T_i)\) and \(Succ(T_i)\) to denote the set of immediate predecessors and successors of task \(T_i\). That is,

\[
\begin{align*}
Pre(T_i) &= \{T_j | (T_j, T_i) \in D\}, \\
Succ(T_i) &= \{T_j | (T_i, T_j) \in D\}.
\end{align*}
\]

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

\[
Pre(T_{entry}) = \emptyset,
\]

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

\[
Succ(T_{exit}) = \emptyset.
\]

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
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 \( I = \bigcup_{i=1}^{\infty} I_i \) denote all the available instances in an IaaS platform, which means the number of instances that a client can access is unlimited. Let \( R = \bigcup_{i=1}^{\infty} R_i \) denote all the instance types where \( m \) is the number of the types. Then we can use \( R_i \), Type\( (I_i) \) to indicate the relation that instance \( I_i \) belongs to type \( R_i \). 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 \( CU(I_i) \) and \( RET(I_i) \) be the compute unit and reference execution time of task \( T \) running on instance \( I_i \), respectively, the actual execution time \( AET(I_i) \) of task \( T \) running on instance \( I_i \) is then formulated as

\[
AET(I_i) = \frac{RET(I_i)}{CU(I_i)}
\]

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 \( T_1, T_2, T_3 \), the data transfer between \( T_2, T_3 \) 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 \( BW(R_i) \) represent the bandwidth of instance type \( R_i \), and \( DATA(T_i, T_j) \) represent the size of data transferred from \( T_i \) to \( T_j \). Supposing tasks \( T_i \) and \( T_j \) are running on instances \( I_i \) and \( I_j \), respectively, the communication time \( CMT(T_i, T_j) \) between tasks \( T_i \) and \( T_j \) is then described in Eq. (6).

\[
CMT(T_i, T_j) = \begin{cases} \min \left\{ \frac{DATA(T_i, T_j)}{BW(\text{Type}(I_i))}, \frac{DATA(T_i, T_j)}{BW(\text{Type}(I_j))} \right\} & I_i \neq I_j, \\ 0, & I_i = I_j. \end{cases}
\]

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 \( ST(T_i) \) and \( FT(T_i) \) 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 \( Pre(T_i) \), the communication time \( CMT(T_i, T_j) \) between its predecessors and itself, and the finish time \( FT(T_i) \) of the previous task that has been executed on the same instance, the finish time \( FT(T_i) \) of task \( T_i \) is calculated as

\[
FT(T_i) = ST(T_i) + AET(I_i) = \max\{AVA(Ins(T_i)), \max_{T_j \in Pre(T_i)} (FT(T_j) + CMT(T_j, T_i))\} + AET(I_i).
\]

where \( AVA(Ins(T_i)) \) is the available time of instance \( Ins(T_i) \) 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., \( FT(T_{entry}) = 0 \). The makespan defined as the finish time of exit task \( T_{exit} \) is then formulated as

\[
\text{Makespan} = FT(T_{exit}).
\]

where \( FT(T_{exit}) \) 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 \( MC(PC_h, R_i, I_i) \) denote the monetary cost of running instance \( I_i \) with type \( R_i \) using pricing model \( PC_h \), where \( PC_h \in PC \), \( 1 \leq h \leq H \). 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

\[
\text{Cost} = \sum_{I_i \in I^*} MC(PC_h, \text{Type}(I_i), I_i)
\]

\[
= \sum_{I_i \in I^*} MC(PC_h, R_i, I_i).
\]

where \( I^* = \{I_i | \exists T_i \in T : \text{Ins}(T_i) = I_i\} \) 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
obstacles which need to be optimized simultaneously:

\[
\text{Minimize } F(x) = \left( F_1(x), F_2(x), \ldots, F_r(x) \right)^T
\]

Subject to \( x \in X \)  

(10)

where \( X \) is the solution space and \( F_r(x) \) is the \( r \)th 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

\[
\forall p : F_p(x_1) \leq F_p(x_2) \land \exists q : F_q(x_1) < F_q(x_2).
\]

(11)

A solution \( x^{\text{opt}} \) 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 \( x_6, x_7, x_8 \) 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 trade-off 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 \( \Psi \subset \mathbb{R}^M \) be the solution space that contains all the possible solution vectors where \( r \) 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 \( \mu^{\text{dom}}_r \) 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

\[
\mu^{\text{dom}}_r(f_r(v) - f_r(u)) = \mu^{\text{dom}}_r(u \succ^r F v).
\]

(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 \( v \succ^r F u \) 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 \( u \succ^F v \) for each \( r \) as

\[
\mu^{\text{dom}}(u \succ^F v) = \bigcap_{r=1}^{M} \mu^{\text{dom}}_r(u \succ^F v),
\]

(13)

where \( \bigcap \) is the fuzzy intersection operation and \( \mu^{\text{dom}}_r(u \succ^F v) \) 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.,

\[
\mu^{\text{dom}}(S \succ^F v) = \bigcup_{u \in S} \mu^{\text{dom}}(u \succ^F v),
\]

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

$$\mu^\text{dom}_{r}(\Delta_f) = \begin{cases} 0, & \Delta_f \leq 0 \\ \frac{\Delta_f}{\Delta_r}, & 0 < \Delta_f < \Delta_r \\ 1, & \Delta_f \geq \Delta_r \end{cases}$$

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

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 prioritization 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 $S = \bigcup_{i=1}^{n} S_i$ denote the set of $K$ selected tradeoff solutions, where $S_i = \bigcup_{i=1}^{K} \{T_i, I_i, Type(I_i)\}$ is the $i$th 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 $I = \bigcup_{i=1}^{n} I_i$. The $n$ instances can be any type of $R = \bigcup_{i=1}^{m} R_i$. Similar to HEFT, the algorithm first determines the scheduling order of tasks in set $T$ by $Rank \leftarrow \text{UpRank}(T)$ (line 1), where $\text{UpRank}()$ 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 $S \leftarrow \emptyset$ and $S_i \leftarrow \emptyset$ (line 2), where $S_i$ 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 $I_i$, the algorithm needs to check whether the type of instance $I_i$ is determined. If the type of instance $I_i$ has not been decided, the solution can be generated by binding $I_i$ to an arbitrary type, then $S_i'$ and $S_i'$ are hence updated (lines 11–14). Otherwise, the solution is directly generated when task $T_i$ is assigned to instance $I_i$ with a fixed type, then $S_i'$ and $S_i'$ are also updated (lines 7–9). After obtaining all the possible solutions, the algorithm sorts these solutions based on their fuzzy dominance values by $S' \leftarrow \text{FuzzyDomianceSort}(S')$, and chooses the first $K$ solutions in $S'$ to form $S$ by $S \leftarrow \text{SelectFirstKItems}(S', K)$ (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 $S = \bigcup_{i=1}^{K} S_i$ where $S_i = \bigcup_{i=1}^{n} \{T_i, I_i, Type(I_i)\}$ (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., $S' = \{S_1, S_2\} = \{(\{T_1, I_1, R_1\}, \{T_1, I_1, R_2\}\}, \{(T_1, I_1, R_2), (T_2, I_2, R_1)\}, \{(T_1, I_1, R_1), (T_2, I_2, R_2)\})$. Both solutions are selected and stored in the solution set $S$ using $S' \leftarrow \text{FuzzyDomianceSort}(S')$ and $S \leftarrow \text{SelectFirstKItems}(S', K)$.

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 $S_1$, $S_2$, and $S_3$ 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 $S_1$, $S_2$, $S_3$, and $S_4$ are selected from $S'$ and stored in set $S$. As a result, $S = \{S_1, S_2\} = \{(T_2, I_2, R_1), (T_1, I_2, R_2), (T_1, I_2, R_1), (T_2, I_2, R_2)\}$. 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 FuzzyDominanceSort() 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.

Algorithm 1: Fuzzy Dominance Sort Based HEFT

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$Rank \leftarrow \text{UpRank}(T)$;</td>
</tr>
<tr>
<td>2</td>
<td>$S \leftarrow \emptyset$;</td>
</tr>
<tr>
<td>3</td>
<td>for $i \leftarrow 1$ to $n$ do</td>
</tr>
<tr>
<td>4</td>
<td>$S_i' \leftarrow \emptyset$;</td>
</tr>
<tr>
<td>5</td>
<td>for $r \leftarrow 1$ to $\text{len}(S)$ do</td>
</tr>
<tr>
<td>6</td>
<td>if type of instance $I_i$ in solution $S_i$ has been decided then</td>
</tr>
<tr>
<td>7</td>
<td>$S_i' \leftarrow S_i' \cup {\text{Rank}_i, I_i, \text{Type}(I_i)}$;</td>
</tr>
<tr>
<td>8</td>
<td>$S' \leftarrow S' \cup {S_i'}$;</td>
</tr>
<tr>
<td>9</td>
<td>else</td>
</tr>
<tr>
<td>10</td>
<td>for $k \leftarrow 1$ to $m$ do</td>
</tr>
<tr>
<td>11</td>
<td>$\text{Type}(I_i) \leftarrow R_k$;</td>
</tr>
<tr>
<td>12</td>
<td>$S_i' \leftarrow S_i' \cup {\text{Rank}_i, I_i, \text{Type}(I_i)}$;</td>
</tr>
<tr>
<td>13</td>
<td>$S' \leftarrow S' \cup {S_i'}$;</td>
</tr>
<tr>
<td>14</td>
<td>break;</td>
</tr>
<tr>
<td>15</td>
<td>$S' \leftarrow \text{FuzzyDominanceSort}(S')$;</td>
</tr>
<tr>
<td>16</td>
<td>$S \leftarrow \text{SelectFirstKItems}(S', K)$;</td>
</tr>
<tr>
<td>17</td>
<td>return $S = \bigcup_{i=1}^{K} S_i$;</td>
</tr>
</tbody>
</table>
According to Eq. (13), calculate \( \mu \) by
\[
\mu \leftarrow \mu \cap \mu^{dom}(S_j) = f_r(S_j); \\
\mu^{dom}(S \succ^F S_j) \leftarrow \kappa; 
\]
and then iteratively derive the fuzzy \( \succ^F \)-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 \( \kappa \) to store the value of fuzzy dominance in a solution set represented by \( \mu^{dom}(S \succ^F S_j) \) (lines 2 and 12). To derive the fuzzy dominance in a solution set using Eq. (14), it needs to calculate the fuzzy \( \succ^F \)-dominance by a solution with 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 \( S_i \) in set \( S \), the perimeter value of \( S_i \) is
\[
P(S_i) = \sum_{r=1}^{M} f_r(u) - f_r(v) \
\max f_r - \min f_r
\]
where \( M \) is the number of objective functions, and \( u \) and \( v \) are solutions that are adjacent to solution \( S_i \) with the same fuzzy dominance value. \( f_r(u) \) and \( f_r(v) \) are the values of \( r \)th objective of solutions \( u \) and \( v \), respectively. \( \max f_r \) and \( \min f_r \) are the maximal and minimal values of \( r \)th objective of all solutions in set \( S \).

### Algorithm 2: Fuzzy Dominance Value Calculation

**Input:** Solution set \( S \);

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

### Algorithm 3: Perimeter Value Assignment

**Input:** Solution set \( S \);

1. \( L \leftarrow \text{len}(S); \)
2. for \( \ell \leftarrow 1 \) to \( L \) do
   3. \( P(S_i) \leftarrow 0; \)
   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. \( P(S_i) \leftarrow \infty; \)
      7. \( P(S_i) \leftarrow \infty; \)
      8. for \( \ell \leftarrow 2 \) to \( (L - 1) \) do
         9. \( P(S_i) \leftarrow P(S_i) + S_{\ell} - S_{\ell-1}; \)
            10. // derive \( P(S_i) \) 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.

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

boundary solutions $S_1$ and $S_2$ 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].

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 $\varepsilon$-Fuzzy PSO [11], MOHEFT [13], NSPSO [10], and SPEA2* [12]. These comparative algorithms are described below.

- $\varepsilon$-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
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 30, 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

\[
\text{Diff(ε-Fuzzy PSO)} = \frac{\text{HV(FDHEFT)}}{\text{HV(ε-Fuzzy PSO)}} \times 100\% - 1.
\]

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

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.

There is no clear conclusion from the table showing why the HV improvements vary significantly between different workflows. However, it is evident that the HV improvements are generally higher for workflows with larger numbers of nodes, as shown in Table 4. This suggests that our FDHEFT is more effective in handling workflows with larger scale.
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 worse 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


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 Orara 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.