IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 28, 2017, accepted June 9, 2017, date of publication June 22, 2017, date of current version July 7, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2715279

Greedy-Ant: Ant Colony System-Inspired Workflow Scheduling for Heterogeneous Computing

BIN XIANG, BIBO ZHANG, AND LIN ZHANG
Beijing University of Posts and Telecommunications, Beijing 100876, China
Corresponding author: Bin Xiang (bingo6@bupt.edu.cn)

This work was supported by the National Key R&D Program of China under Grant 2016YFB0100902.

ABSTRACT The last decades have seen a considerable progress on workflow scheduling in heterogeneous computing environments. However, existing methods still need to be improved on the performance in the makespan-based metrics. This paper proposes a novel workflow scheduling algorithm named *Greedy-Ant* to minimize total execution time of an application in heterogeneous environments. First, the ant colony system is applied to scheduling from a new standpoint by guiding ants to explore task priorities and simultaneously assign tasks to machines. Second, *forward/backward dependence* is defined to indicate the global significance of each node, based on which, a new heuristic factor is proposed to help ants search for task sequences. Finally, a greedy machine allocating strategy is presented. Experimental results demonstrate that *Greedy-Ant* outperforms the state of the art up to 18% in the metric of speedup.

INDEX TERMS Workflow scheduling, makespan, heterogeneous computing, ant colony system.

I. INTRODUCTION

Heterogeneous computing environments provide scalable computing resources for various applications, which is constructed by interconnecting machines with distinct processing capacity via different networks. Workflow scheduling in heterogeneous computing environments aims at assigning tasks to machines to achieve highly efficient computing. In practice, there are numerous resources in the environments and many tasks to be scheduled. Thus workflow scheduling is one of the key challenges to the performance enhancement of heterogeneous computing.

Workflow scheduling has been proved to be an NP-hard optimization problem [1]. There are various studies on how to schedule tasks onto servers. Metaheuristic optimization methods such as Ant Colony Optimization (ACO) [2]–[4], Simulated Annealing (SA) [5], [6], Particle Swarm Optimization (PSO) [7], Genetic Algorithm (GA) [8], Cat Swarm Optimization (CSO) [9], etc., are largely introduced in workflow scheduling. Heuristic-based scheduling methods typically include HEFT [10] and PEFT [11].

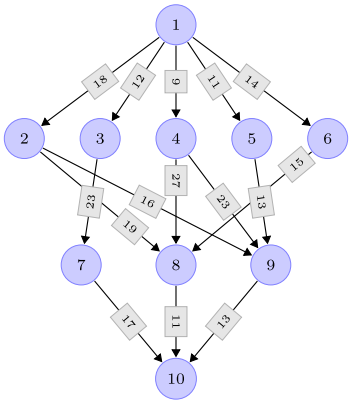
Tawfeek *et al.* [2] proposed an ACO-based task scheduling method to minimize the makespan of an application in cloud environments. Chen and Zhang [3] exploited ACO to schedule large-scale workflows with various QoS parameters. FATS [12] translated the scheduling issue into a TSP-likened problem so that it can be solved by ACO algorithm. However, most of these methods do not make full use of task priorities. Their task sequences are generated by randomly and successively choosing tasks that satisfy precedence constraints. To perform constrained workflow scheduling, Kianpisheh *et al.* [4] proposed the probability of violation (POV) of run-time constraints as a criterion for the schedule robustness, and utilized an ant colony system to minimize an aggregation of violation of constraints and the POV. SA-based scheduling algorithms [5], [6] were proposed to handle scheduling problems in grid computing and cloud computing by applying practical annealing rules to the optimization process of task scheduling. PSO-based algorithms denote the number of tasks by the dimension of particles and let the positions of particles represent the correspondence between virtual machines and tasks [7]. In GA-based scheduling schemes, a mapping from tasks to virtual machines is represented as a single gene in the valid chromosome. The gene order exhibits the schedule execution order on the selected machines. CSO was introduced in [9] and operated in seeking mode and tracing mode to complete task scheduling.

HEFT [10] algorithm, typically belonging to list-based heuristic methods, selects the task with the highest upward rank value and the corresponding processor. PEFT [11] defines optimistic cost table (OCT) and outperforms HEFT

in some cases, but its performance is not robust facing graphs with complex structures. Since these heuristic methods only determine the task sequence at the beginning of the algorithm and do not change it afterwards, the scheduling results tend to be trapped in a local optima.

The goals of workflow scheduling vary in different computing environments. If we consider the general cloud environment such as the public and private cloud, there may be many QoS related metrics to be involved or targets to be optimized. For instance, if a commercial provider is in consideration, at least the economic costs of executions should be taken into the optimization problem. Note that this paper only focuses on scheduling static workflows in a pure heterogeneous computing environment. In a static workflow, all the information of tasks such as the computation and communication costs and the task graph structure are known in advance.

The motivation behind our study is to apply Ant Colony System (ACS) to workflow scheduling to obtain higher quality schedules for heterogeneous computing. ACS makes full use of the instance-based heuristic information, which is discovered to be essential to scheduling problems. Therefore, the proposed *Greedy-Ant* takes advantage of ACS theory to perform workflow scheduling. Unlike most of the previous work [2], [3] making ant colony only search for machine allocation for tasks, *Greedy-Ant* successively seeks tasks according to their significances and simultaneously looks for the most suitable machine for each task. Each time when the ant colony completes searching once, the best schedule is selected to generate a feedback for the ant colony via pheromone. After several times of searching by the colony, a high-quality scheduling result can be found.



Computation Cost.

Task	P1	P2	P3
1	14	16	9
2	13	19	18
3	11	13	19
4	13	8	17
5	12	13	10
6	13	16	9
7	7	15	11
8	5	11	14
9	18	12	20
10	21	7	16

FIGURE 1. A task DAG example. P1, P2 and P3 are the processors. The nodes represent tasks and the edges indicate the dependency of tasks. The weights on the edges denote the communication costs. The table shows the computation costs of tasks on the machines.

Generally, an application can be represented as a directed acyclic graph (DAG). A typical task DAG is shown in Figure 1. Based on this DAG, we show an example of comparison between the results of some existing scheduling methods [2], [10] and *Greedy-Ant* in Figure 2. It is shown

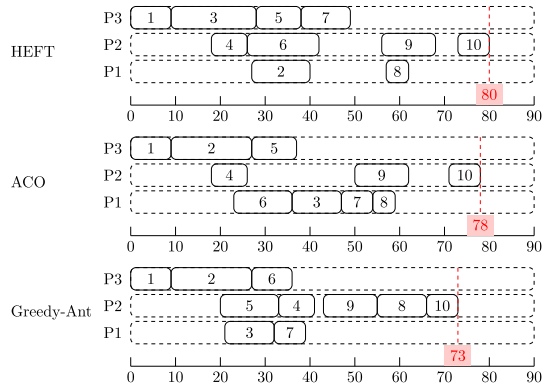


FIGURE 2. A comparison between different methods on scheduling the graph in Figure 1. The vertical axis represents the machine assignment, while the horizontal axis indicates the order and execution time. The makespan of tasks scheduled by HEFT [10] is 80. The ACO [2] allows the makespan of tasks to be 78. *Greedy-Ant* reduces the makespan to 73.

that the scheduling scheme generated by *Greedy-Ant* has the minimum makespan.

Our contributions are summarized as follows:

- A new perspective is provided on how to apply ACS to workflow scheduling. Exactly, the ant colony not only applies priority-based searching scheme to produce task sequences but also performs low-complexity greedy machine allocation.
- A new heuristic information is developed to form the transition probability of ants to generate searching paths during scheduling, based on the proposed *forward dependency* and *backward dependency* definitions.
- Compared with state-of-the-art approaches, *Greedy-Ant* significantly improves the scheduling quality in metrics of makespan, speedup, schedule length ratio and frequency of better results, especially in high-concurrency and high-heterogeneity environments.

The remainder of this paper is organized as follows. Section II introduces the workflow scheduling problem. Section III presents the proposed mathematical models. In section IV, we demonstrate the performance of *Greedy-Ant* via several experiments and discuss the limitations. Finally, section V summarizes our work and describes some of our future directions.

II. PROBLEM STATEMENT

In this paper, $G(\mathcal{V}, \mathcal{E})$ represents a task DAG. Each node $v_i \in \mathcal{V}$ indicates a task. Each directed edge $e(i, j) \in \mathcal{E}$ represents dependency constraint between v_i and v_j such that v_i should be completed before v_j can be started.

In addition, the $n \times m$ computation matrix $\boldsymbol{W}$ stores the execution costs of tasks $\mathcal{V}$ running on machines $\mathcal{M}$. $n = |\mathcal{V}|$ is the number of tasks and $m = |\mathcal{M}|$ represents that of machines. The element $w_{i,t}$ is the execution time of task v_i on machine m_t. In a task graph, $e(i, j)$ is associated with a weight $c_{i,j}$ which represents the communication time between v_i and v_j. Note that when v_i and v_j are assigned to the same machine, $c_{i,j}$ equals zero since it is negligible when compared with interprocessor communication cost.

In a task graph, a node with no predecessors is called an entry task v_{entry}, and a node with no successors is called an exit task v_{exit}. In practice, a pseudo entry node and an exit node with zero-cost in computation and communication are added to unify descriptions of a task graph. In other words, the pseudo entry is the immediate predecessor of the original entries and the pseudo exit is the immediate successor of the original exits. This addition will not affect scheduling.

We list the basic assumptions of the computing environment as follows.

- All the machines are fully connected.
- Task execution and data communication between machines can be at the same time.
- Task execution is nonpreemptive on each machine.

These assumptions are derived from general real systems, which ensure a fair comparison with state-of-the-art scheduling algorithms.

We present the following definitions for the scheduling problem.

Definition 1: For a task node v_i in a given task graph, $pred(v_i)$ is the set of its immediate predecessors and $succ(v_i)$ the set of its immediate successors.

Definition 2: For v_i which has already been scheduled on a machine, $AST(v_i)$ is its Actual Start Time (AST) and $AFT(v_i)$ its Actual Finish Time (AFT).

Definition 3: $T_{ready}(v_i, m_t)$ represents the time when v_i is ready to be scheduled on a machine m_t. If v_i is an entry node, $T_{ready}(v_{entry}, m_t) = 0$, otherwise it is defined as

$$T_{ready}(v_i, m_t) = \max_{v_k \in pred(v_i)} \{AFT(v_k) + c_{k,i}\}. \quad (1)$$

Definition 4: $EST(v_i, m_t)$ indicates the Earliest Start Time (EST) of v_i on m_t and is defined as

$$EST(v_i, m_t) = \max\{T_{ready}(v_i, m_t), T_{avail}(m_t)\}, \quad (2)$$

where $T_{avail}(m_t)$ is the time when m_t is available for receiving a task.

Definition 5: $EFT(v_i, m_t)$ denotes the Earliest Finish Time (EFT) of v_i on m_t and is defined as

$$EFT(v_i, m_t) = EST(v_i, m_t) + w_{i,t}. \quad (3)$$

Definition 6: *makespan* represents the Actual Finish Time of the exit task v_{exit} and is defined as

$$makespan = AFT(v_{exit}). \quad (4)$$

One of our aims is to assign tasks in a given task graph to proper machines to minimize *makespan*.

III. GREEDY-ANT

Greedy-Ant generally operates in two phases: (1) *task prioritizing* and (2) *machine allocating*. In the first phase, the ant colony is employed to generate task sequences. A new heuristic information considering global dependency of a task node is proposed so that ants can find better task priorities and the searching process can speed up. In the second phase, a greedy minimum strategy is presented for the ant colony to seek the best machine in each round of searching.

Ant colony system is a metaheuristic method inspired by the foraging behavior of real ant colonies [13]. When a group of ants tries to search for food, they use a special chemical called pheromone to communicate with each other. Each moving ant will deposit some pheromone along a path. An ant will follow the trail containing the most pheromone with the highest probability. As the foraging process continues, the ants tend to choose the shortest path which has accumulated a large amount of pheromone.

A searching strategy for ant colony is needed for ants to find a pleasant solution. First of all, we assume that valid edges and paths should observe the following rules.

- Edge $e(v_i, v_j)$ should merely include direct single-hop connection from v_i to v_j;
- Path $(v_i \rightarrow v_j)$ can include both direct and indirect connections satisfying precedence constraints.

A. NEW HEURISTIC INFORMATION

The heuristic information and the pheromone are the most important factors in ACS based methods. In general, heuristic information is a problem-based parameter to guide the moving direction of an ant, which can accelerate the convergence of the algorithm. The pheromone records the trails that can influence the subsequent searching behaviors of ants. The ant colony searches for the best path based on these two parameters to determine the scheduling scheme.

In order to reflect the dependence of a task node v upon all its predecessors, the *forward dependency* function of v is defined as

$$FD(v) = \begin{cases} \sum_{v_p \in pred(v)} FD(v_p), & if\ pred(v) \neq \emptyset, \\ 1, & otherwise. \end{cases} \quad (5)$$

Similarly, to indicate how important v is to all its successors, the *backward dependency* function of v is defined as

$$BD(v) = \begin{cases} \sum_{v_p \in succ(v)} BD(v_p), & if\ succ(v) \neq \emptyset, \\ 1, & otherwise. \end{cases} \quad (6)$$

$FD(\cdot)$ and $BD(\cdot)$ are computed in a recursive manner.

Online forward/backward dependency function is defined to be the sum of forward/backward dependency relations of all the unvisited predecessors/successors of v_j when an ant arrives at v_i:

$$f_{FD}(v_j) = \sum_{v \in \mathcal{J}_{ij}} FD(v), \quad (7)$$

$$f_{BD}(v_j) = \sum_{v \in succ(v_j)} BD(v), \quad (8)$$

where $\mathcal{J}_{ij} = \{v | v \in pred(v_j) \cap \mathcal{S}_i\}$ denotes the set of the unvisited immediate predecessors of v_j when an ant arrives at v_i. $\mathcal{S}_i$ is the set of the unvisited nodes.

In a given task graph, different nodes have different execution priorities. A node with many predecessors may have lower priority since it can be started only if all its predecessors

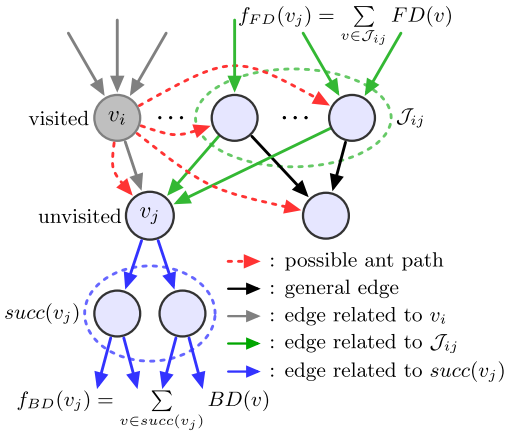


FIGURE 3. Heuristic information. There are some possible paths for the current ant to move. The importance of the candidate v_j is measured based on $FD(v_j)$ and $BD(v_j)$.

are completed. In the meanwhile, a node with many successors may have higher priority since its successors can be executed only if it is completed. Based on the above dependency relations among tasks (see Fig. 3), this paper proposes the heuristic information on path $(v_i \to v_j)$ as follows:

$$\eta_{ij} = \begin{cases} \dfrac{f_{BD}(v_j)}{f_{FD}(v_j)}, & if\ v_i \to v_j\ exists, \\ 0, & otherwise. \end{cases} \tag{9}$$

B. TRANSITION RULE OF THE ACS

The transition rule [13] followed by the kth ant is

$$v_{next} = \begin{cases} \underset{v_j \in C_k(v_i)}{\text{argmax}}\{[\tau_{ij}][\eta_{ij}]^\alpha\}, & if\ r \leqslant r_0, \\ v_n, & otherwise, \end{cases} \tag{10}$$

where $C_k(v_i)$ is the set of candidate nodes for the next move of the kth ant, τ_{ij} represents the amount of pheromone on path $(v_i \to v_j)$ and α controls the importance of η_{ij}. r is a random number uniformly distributed in [0, 1] and r_0 is a parameter $(0 \leqslant r_0 \leqslant 1)$. v_n is the most likely next move randomly selected according to the probability:

$$p_{ij}^k = \begin{cases} \dfrac{[\tau_{ij}][\eta_{ij}]^\alpha}{\sum_{v_c \in C_k(v_i)}[\tau_{ic}][\eta_{ic}]^\alpha}, & if\ v_j \in C_k(v_i), \\ 0, & otherwise. \end{cases} \tag{11}$$

C. PHEROMONE UPDATING

During foraging, ants pass through some edges and change the pheromone by:

$$\tau_{ij} = (1 - \xi)\tau_{ij} + \xi\tau_0, \tag{12}$$

where τ_0 is an initial pheromone level and ξ is a parameter $(0 < \xi < 1)$. In order to decay the pheromone to make the visited paths less desirable, τ_0 is usually small. $\tau_0 = 1/(L_{rand} \cdot |\mathcal{V}|)$ is set by default, where L_{rand} is the *makespan* of a random schedule. When one round of searching is completed, the global best ant (i.e., the ant which constructs the shortest tour) is allowed to update pheromone on the path $(v_i \to v_j)$:

$$\tau_{ij} = (1 - \rho)\tau_{ij} + \rho\Delta\tau_{ij}, \tag{13}$$

where ρ is the pheromone decay parameter. $\Delta\tau_{ij} = 1/L_{best}$ if $(v_i \to v_j)$ belongs to the best tour, otherwise $\Delta\tau_{ij} = 0$. L_{best} is the minimum time cost found by the best ant.

D. GREEDY-ANT

The basic idea of *Greedy-Ant* is elaborated in Algorithm 1. Firstly, *FD*, *BD* and the initial pheromone matrix $\boldsymbol{\tau}$ are computed for each node. Then the ant colony is employed to successively search for the best schedule for $iter_{max}$ times. Each ant generates a task sequence, allocates machines to run these tasks and locally updates the pheromone. When all the ants finish searching, the global best ant performs global pheromone updating. The best assignment corresponding to the minimum *makespan* is recorded.

Algorithm 1 *Greedy-Ant*

```
1:  iter = 0;
2:  Initialize pheromone [τ_ij^iter];
3:  Compute FD and BD value using Eq. (5) & (6);
4:  while iter < iter_max do
5:      for k ← 1 to K do
6:          list ← {v_entry}; S^k ← ∅;
7:          while list ≠ ∅ do
8:              Compute [η_ij] using Eq. (9);
9:              Compute p based on Eq. (11);
10:             // Normalize the probability of v_s → v_t.
11:             P_RWS ← {p_st / Σ_{v_t ∈ list} p_st};
12:             Select v_next from list via RWS method;
13:             Update local trail using Eq. (12);
14:             S^k ← {S^k, v_next}; // Record visited node.
15:             list ← list − v_next; // Remove v_next.
16:             for all v_t ∈ succ(v_next) do
17:                 if pred(v_t) are all visited then
18:                     list ← {list, v_t}; // Update list.
19:         Compute EFT^k via greedy minimum strategy;
20:         Allocate m_t ∈ M to v_i ∈ S^k w.r.t. EFT^k_min;
21:     Record best schedule w.r.t. min{makespan};
22:     Update global trail using Eq. (13);
23:     iter ← iter + 1;
```

Exactly, Lines 6-18 show the *task prioritizing* phase for the kth ant. The heuristic information matrix $\boldsymbol{\eta}$ is dynamically updated. The transition probability matrix $\boldsymbol{p}$ is then computed combining $\boldsymbol{\tau}$ and $\boldsymbol{\eta}$. A list is constructed by initially inserting v_{entry} and gradually adding the tasks which satisfy precedence constraints. To this end, for the successors of the latest v_i in the sequence, if all of their predecessors have been executed (i.e., in the task sequence), they will be added to the list. To gradually build a task sequence S^k, *Greedy-Ant* uses Roulette Wheel Selection (RWS) method to select a task from

the list and adds it to the sequence. The roulette wheel is built based on the normalized transition probabilities P_{RWS}.

The basic idea behind our *machine allocating* phase (Lines 19-20) is to let the kth ant greedily look for suitable machines for tasks. For v_i in S^k, a greedy minimum strategy is adopted to select a machine. Exactly, the Earliest Finish Time $EFT(v_i, m_t)$ is computed and m_t corresponding to the minimum (EFT^k_{min}) is allocated to execute v_i.

Note that the differences between *Greedy-Ant* and the existing ACO-based workflow scheduling techniques lie in two aspects. a) *Greedy-Ant* provides a new idea that an ant colony is employed to search task sequences and simultaneously allocate machines by a greedy policy, while other ACO-based methods fix the task sequence initialized randomly and use ACO to search the machine allocating schemes. b) *Greedy-Ant* defines a new heuristic based on *FD*/*BD* to facilitate searching, while other ACO-based methods compute the inverse computation and communication costs as the heuristic.

E. COMPLEXITY OF GREEDY-ANT

Greedy-Ant mainly operates in two phases: task prioritizing and machine allocating (see Algorithm 1). In each iteration, each ant searches for a feasible task sequence. This process has an $\mathcal{O}(n + e)$ time complexity for n nodes and e edges. For a dense graph, the number of edges is proportional to $\mathcal{O}(n^2)$, so that the time complexity in the worst case becomes $\mathcal{O}(n^2)$. Upon obtaining a task sequence, a greedy strategy searching for the minimum EFT (EFT^k_{min}) is performed to allocate machines to the tasks in order. The time complexity of this process is $\mathcal{O}(nm)$, where m is the number of machines. The total time complexity of one iteration is $\mathcal{O}(K(nm + e))$ and $\mathcal{O}(K(nm + n^2))$ in the worst case.

IV. EXPERIMENTAL RESULTS AND DISCUSSION

In this section, the performance of *Greedy-Ant* is compared with state-of-the-art scheduling methods including PEFT [11], HEFT [10], SA [5] and a typical ACO-based scheduling strategy [2]. Despite different means of applying ACO, the adopted strategies in the literatures are similar. Besides, the algorithm proposed in [2] is aimed to merely minimize makespan, which is consistent with the goal of *Greedy-Ant*, while [3] and [4] focus on solving the constrained scheduling problems. Therefore, the scheduling method [2] is chosen to be compared with *Greedy-Ant*. *Greedy-Ant* is implemented using Matlab R2013a platform and tested on two sets of task graphs: randomly generated application graphs and real-world application graphs. Before analyzing the experimental results, the parameter settings for *Greedy-Ant* and the metrics used for performance comparison are listed below.

A. PARAMETERS FOR GREEDY-ANT

Greedy-Ant adopts the following parameters.

- The maximum iteration in ACS is $iter_{max} = 5000$;
- There are $K = 50$ ants in the ant colony;
- The global pheromone decay parameter is $\rho = 0.1$;
- The local pheromone decay parameter is $\xi = 0.1$;
- The importance of heuristic information is $\alpha = 1$;
- r_0 in Equation (10) meets $r_0 = 0.9$;

B. COMPARISON METRICS

The comparisons are performed based on the metrics of schedule length ratio (SLR), speedup and frequency of better results.

1) SCHEDULE LENGTH RATIO (SLR)

A key measurement of a scheduling algorithm is the *makespan* of the schedule obtained. Due to the distinct graph topology, *makespan* needs to be normalized to a lower bound. The schedule length ratio (SLR) [10] is defined as:

$$SLR = \frac{makespan}{\sum_{v_i \in CP_{min}} \min_{m_t \in \mathcal{M}} \{w_{i,t}\}}. \tag{14}$$

The denominator is the summation of the minimum computation costs of tasks on the critical path CP_{min}. For any scheduling algorithms, the SLR value of a graph is larger than one, since the denominator is the lower bound. Therefore, the lower SLR value is, the better performance the algorithm will have.

2) SPEEDUP

The speedup value is defined as the ratio of the sequential execution time to the parallel execution time (i.e., *makespan*):

$$Speedup = \frac{\min_{m_t \in \mathcal{M}} \{\sum_{v_i \in \mathcal{V}} w_{i,t}\}}{makespan}, \tag{15}$$

where $\min_{m_t \in \mathcal{M}} \{\sum_{v_i \in \mathcal{V}} w_{i,t}\}$ is the sequential execution time computed by assigning all tasks to a single machine that minimizes the total computation cost.

3) FREQUENCY OF BETTER RESULTS

The percentage of better, worse, and equal quality solutions produced by *Greedy-Ant* is compared with that of the remaining algorithms.

C. DAG SETTINGS AND COMPUTING INFRASTRUCTURE

Greedy-Ant and the other algorithms are tested on the following DAGs and computing infrastructure. The popular DAG generating program DAGGEN available at [14] is modified to obtain the random and synthetic DAGs used in the experiments. The graph shape is determined by the following parameters.

- n: the number of task nodes in a DAG.
- fat: the parallelism degree of a DAG, given n. A large value results in a shorter DAG with high parallelism, while a small value leads to a longer DAG with low parallelism. Exactly, the maximum number of tasks in each level of a DAG is randomly determined according to a uniform distribution with the mean $fat \cdot \sqrt{n}$. The depth of a DAG is a random value decided by a uniform distribution with the mean $\sqrt{n}/fat$.

- *density*: the dependency degree of the nodes in a graph. The larger *density* value is, the stronger the dependency is.
- *regularity*: the similarity of the task numbers between levels. A large value indicates the high similarity.
- *jump*: the number of levels spanned by communications (i.e., an edge can jump over *jump* levels).

In order to describe the communication and computation costs of a task graph, we adopt the following parameters:

- Communication-to-computation ratio (CCR): the ratio of the average communication cost to the average computation cost.
- β (Range of computation costs on machines): the heterogeneity factor for the performance of machines on speed. A high β value indicates higher heterogeneity and more complex computation costs among processors, and a low value implies that the computation costs for a given task are nearly the same among machines [10]. The communication cost $\overline{w_i}$ of the task v_i in a DAG is randomly set according to a uniform distribution in $[0, 2 \times \overline{w_{DAG}}]$, where $\overline{w_{DAG}}$ is the average computation cost of a given random graph. The computation cost of each task v_i on the machine m_t is randomly selected from $[\overline{w_i}(1-\beta/2), \overline{w_i}(1+\beta/2)]$.

The computing infrastructure is constructed by connecting a certain number of machines with different computing capabilities in a general network. The number of machines is denoted by m. *CCR* and β can indicate the communication and computation heterogeneity of different machines.

D. RANDOM TASK GRAPHS

Firstly, experiments are run on random task graphs. In each comparison, to randomly generate DAGs, we choose the values of the parameters from the corresponding sets below.

- $n = \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400, 500\}$;
- $CCR = \{0.1, 0.2, 0.5, 0.8, 1, 2, 5, 8, 10\}$;
- $\beta = \{0.1, 0.2, 0.5, 1, 2\}$;
- $jump = \{1, 2, 4\}$;
- $regularity = \{0.2, 0.5, 0.8\}$;
- $fat = \{0.1, 0.4, 0.8\}$;
- $density = \{0.2, 0.5, 0.8\}$;
- $m = \{2, 4, 8, 16, 32\}$;

The combinations of parameter settings above generate 255150 different sorts of DAGs. For the DAGs with the same parameter setting, 20 random graphs with distinct communication and computation costs are generated so that actually 5103000 different DAGs are used in our experiments.

At first, based on the random task graphs, an experiment shown in Figure 4 compares *Greedy-Ant* and its counterpart without the proposed heuristic information. It is obvious that the new heuristic information contributes quite a lot to the improvement of the algorithm.

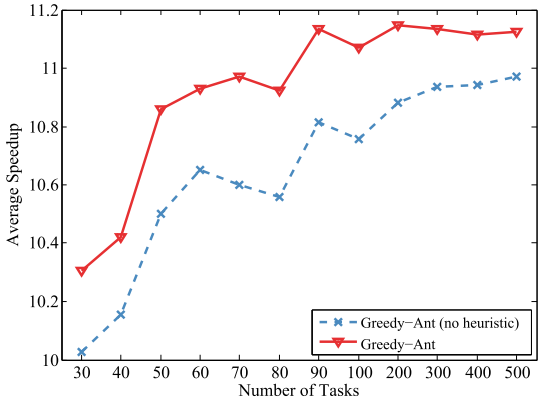


FIGURE 4. Comparison between *Greedy-Ant* and its counterpart without the proposed heuristic information.

Figure 5 shows the performance on speedup and SLR versus the number of tasks, CCR, β and the number of machines for all the algorithms.

1) SPEEDUP AND SLR-NUMBER OF TASKS

In Figure 5(a), *Greedy-Ant* performs better than any other algorithms. Compared with ACO, *Greedy-Ant* improves speedup by about 15 percent when DAGs include 10 tasks. As the task number increases, *Greedy-Ant* gradually shows its advantage. When there are 40 tasks in DAGs, our algorithm can significantly outperforms [2] by 59 percent. It is apparent that the performance of *Greedy-Ant* tends to converge when task number exceeds 50, whereas ACO still witnesses an increase after 100. The performances of PEFT and HEFT are about the same level. Although SA performs better than PEFT and HEFT when $n < 90$, its performance decreases afterwards. When $n = 50$, *Greedy-Ant* improves speedup by about 11 percent, compared with HEFT. For the metric SLR, Figure 5(b) shows the details of the comparisons. It is obvious that *Greedy-Ant* performs best especially in the cases of a large number of tasks.

2) SPEEDUP-β

Figure 5(c) implies that a high degree of heterogeneity will benefit the advantage of *Greed-Ant*. When facing high heterogeneity ($\beta = 2$), *Greedy-Ant* has the significant improvement over SA by 61 percent, ACO by 56 percent, PEFT by 44 percent and HEFT by 13 percent. This demonstrates the effectiveness of *Greedy-Ant* in heterogeneous environments. As heterogeneity decreases to $\beta = 0.5$, this improvement decreases largely. Although this improvement over HEFT, PEFT and SA suffers from a dramatic decrease due to the low heterogeneity of machines, whereas they are still obviously better than ACO.

3) SPEEDUP AND SLR-CCR

We can see from Figure 5(d) that speedup decreases as CCR increases for all the algorithms, implying that small communication-to-computation ratio is beneficial to

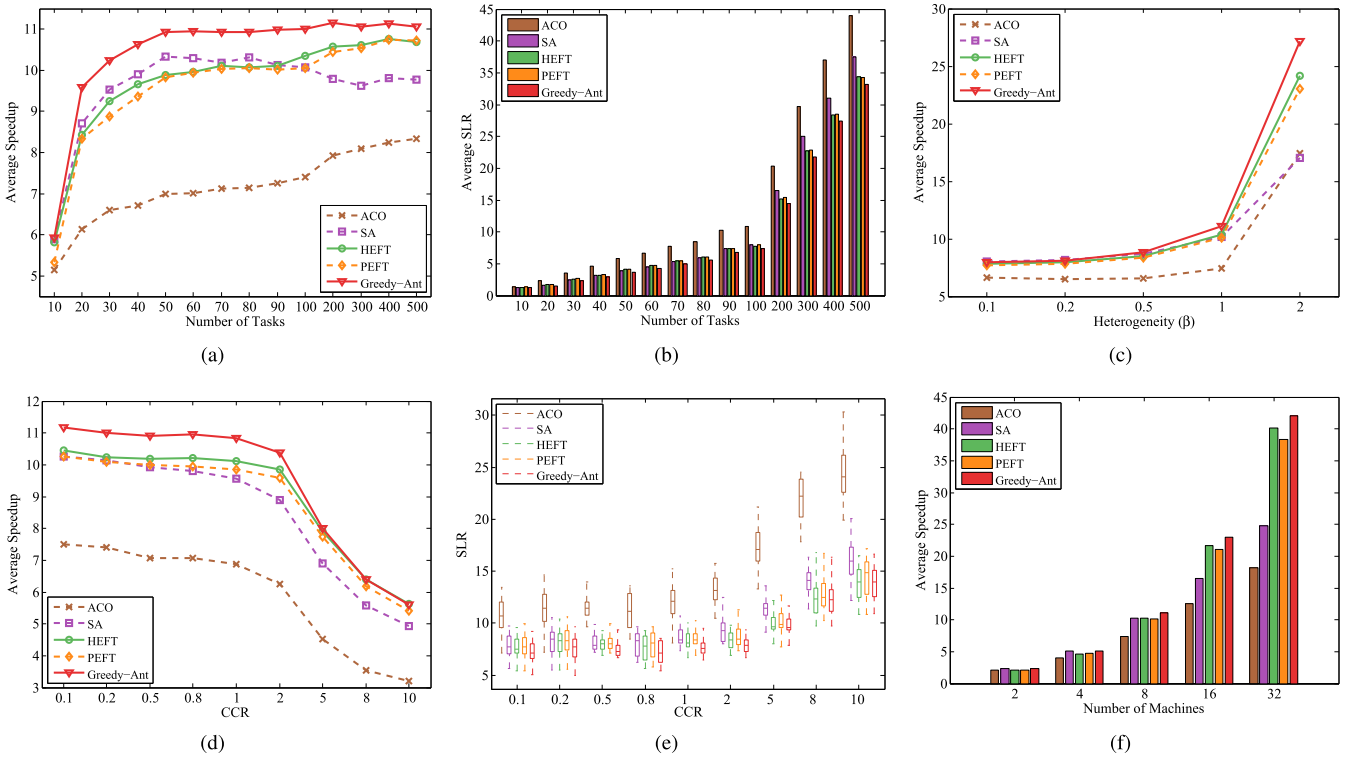


FIGURE 5. Comparison on random task graphs.

task scheduling. As a whole, *Greedy-Ant* outperform the rest of the algorithms. It is interesting to see that when $5 < CCR < 10$ the performance of *Greedy-Ant* almost remains the same on speedup. When CCR is equal to 2, *Greedy-Ant* is better than HEFT by 7 percent, PEFT by 10 percent, SA by 19 percent and ACO by 69 percent.

The SLR computed for *Greedy-Ant*, PEFT, HEFT, SA and ACO as a function of CCR is shown in Figure 5(e). To show the statistical details of the results, we draw boxplots where the minimum, first quartile, median, third quartile and maximum of the SLRs are presented. We can conclude that *Greedy-Ant* always has the lower average SLR and a relative small dispersion. HEFT also performs well.

4) SPEEDUP-NUMBER OF MACHINES

Figure 5(f) demonstrates that the increase of the processor number is beneficial to raise the speedup. At the same time, *Greedy-Ant* always performs better than the other algorithms.

5) FREQUENCY OF BETTER RESULTS

Table 1 is the statistical results of all the comparison experiments on the 5103000 different task DAGs, and illustrates the performance of *Greedy-Ant* with respect to the percentage of DAGs in which it performs better, equally or worse, compared with the remaining algorithms. The cells in Table 1 show the comparison results of the algorithms on the left with those on the top. It is clear to see from the first row that *Greedy-Ant* achieves better scheduling results than the other algorithms in most cases. The results shown in Table 1 and Figure 5 are consistent.

TABLE 1. Pairwise comparison of the scheduling algorithms.

		Greedy-Ant	PEFT	HEFT	SA	ACO
Greedy-Ant	better	*	98.8%	94.8%	87.1%	100.0%
	equal		0.0%	0.0%	2.1%	0.0%
	worse		1.2%	5.2%	10.8%	0.0%
PEFT	better	1.2%	*	30.9%	53.0%	98.3%
	equal	0.0%		0.2%	0.0%	0.0%
	worse	98.8%		68.9%	47.0%	1.7%
HEFT	better	5.2%	68.9%	*	60.5%	99.2%
	equal	0.0%	0.2%		0.0%	0.0%
	worse	94.8%	30.9%		39.5%	0.8%
SA	better	10.8%	47.0%	39.5%	*	98.0%
	equal	2.1%	0.0%	0.0%		0.0%
	worse	87.1%	53.0%	60.5%		2.0%
ACO	better	0.0%	1.7%	0.8%	2.0%	*
	equal	0.0%	0.0%	0.0%	0.0%	
	worse	100.0%	98.3%	99.2%	98.0%	

E. REAL-WORLD APPLICATION GRAPHS

Apart from randomly generated task graphs, we compare the performance of *Greedy-Ant* with state-of-the-art algorithms on the task graphs of some typical scientific real-world applications, including CyberShake workflow [15] and SIPHT

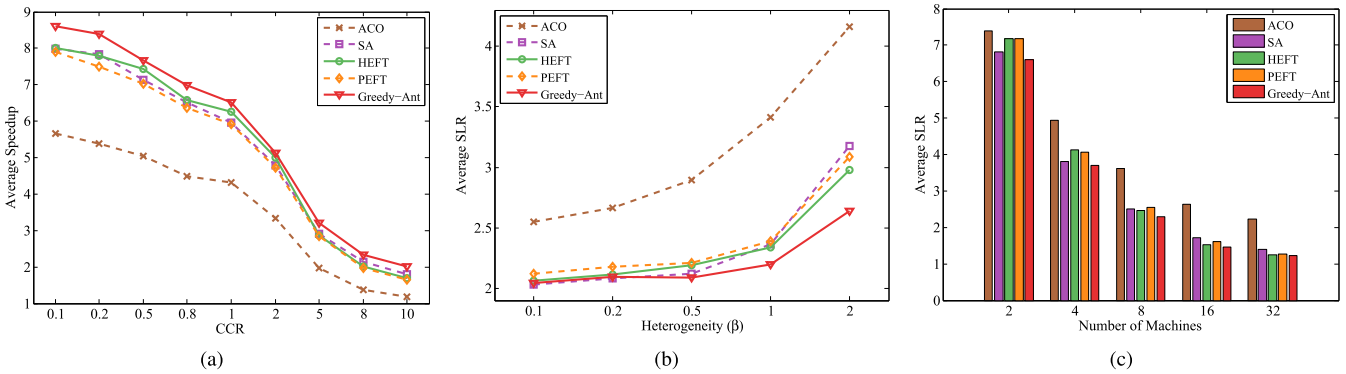


FIGURE 6. Comparison on CyberShake workflow.

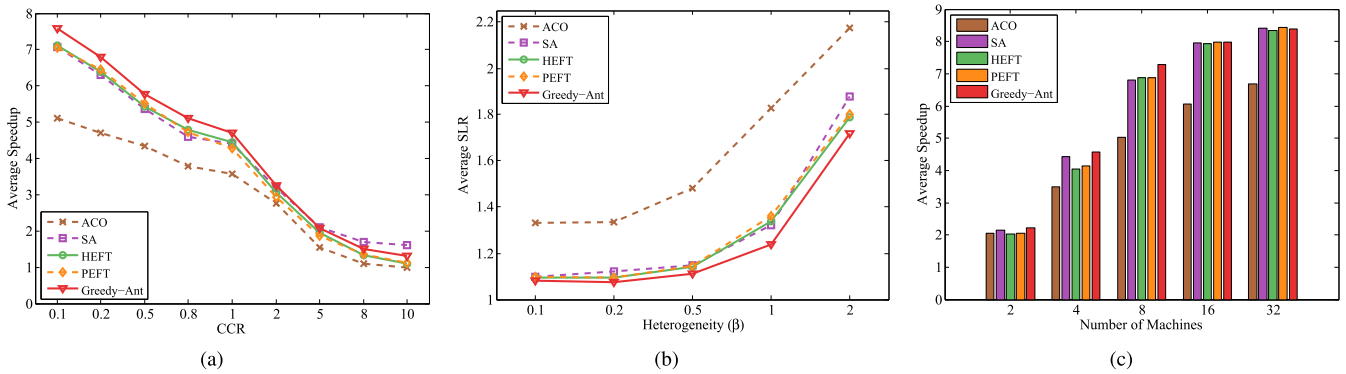


FIGURE 7. Comparison on SIPHT workflow.

workflow [16]. Since the structures of these applications are known, we only consider the performance with respect to CCR, heterogeneity β and the machine number m. CCR in our experiment varies in $\{0.1, 0.2, 0.5, 0.8, 1, 2, 5, 8, 10\}$. β varies in $\{0.1, 0.2, 0.5, 1, 2\}$ and m in $\{2, 4, 8, 16, 32\}$. It is worth noting that CCR and β indicate the properties of machines in the sense that CCR represents the focus (i.e., communication, computation) of a network. β implies the extent of heterogeneity of a machine system.

1) CyberShake WORKFLOW

The CyberShake workflow is known as the tool exploited by the Southern California Earthquake Center to characterize earthquake hazards. We respectively test all the algorithms on the CyberShake workflow with 30 and 50 task nodes. Note that the graph structure for a application when given the fixed number of the tasks. Thus we here compare the algorithms in terms of CCR, β and the number of machines.

Figure 6 shows the average speedup and schedule length ratio as functions of CCR, β and the number of machines. In Figure 6(a) *Greedy-Ant* obviously outperforms PEFT, HEFT and SA when $CCR \leqslant 0.2$. Even if this improvement decreases when $CCR > 0.2$, *Greedy-Ant* still performs better than the other algorithms. In Figure 6(b), when β is relatively large (i.e., high-heterogeneity network), the improvement of *Greedy-Ant* is significant. Concerning the number of machines, Figure 6(c) shows that *Greedy-Ant* can achieve the less schedule length ratio.

2) SIPHT WORKFLOW

The SIPHT workflow in a bioinformatics project at Harvard can automatically search for untranslated RNAs for bacterial replicon in the National Center for Biotechnology Information (NCBI) database. We consider the cases of 30 and 60 task nodes in the SIPHT workflow graph.

Figure 7 shows the performances of all the algorithms on the average speedup and SLR in terms of CCR, β and the number of machines. In Figure 7(a), when $CCR > 5$, SA outperforms *Greedy-Ant*. When $CCR < 5$, the average performances can be roughly sorted from the best and to the worst as *Greedy-Ant*, SA, HEFT, PEFT, ACO. Figure 7(b) shows generally the same trend as Figure 6(b). In Figure 7(c), when $m \leqslant 8$, *Greedy-Ant* clearly outperforms the remaining algorithms. However, when $m = 16, 32$, due to the upper boundary of speedup has been reached, all the algorithms are roughly equal.

F. DISCUSSION

Figure 5, 6, 7, and Table 1 demonstrate that *Greedy-Ant* outperforms HEFT, PEFT, SA, and ACO with respect to the metrics including speedup, SLR, and frequency of better results. HEFT and PEFT are list-based heuristic methods. HEFT defines an indicator $rank_u$ to represent the longest path length from a task node to the exit node. A fixed task sequence is generated by sorting $rank_u$ in descending order. Similarly, PEFT defines $rank_{oct}$ according to an optimistic cost table (OCT) to generate the task sequence. Since the

scheduling problem is NP-complete and the practical combination number of the feasible task sequences is very large, both HEFT and PEFT try to find a local optimum sequence according to a local heuristic indicator. Their performances heavily rely on the effectiveness of the heuristic indicators. As the complexity of the task graph grows, it becomes harder for them to produce consistent results on a variety of graphs. For machine allocating, HEFT uses an insertion-based policy to insert a task in an earliest idle time between two already scheduled tasks on a processor, while PEFT completes it by minimizing a value combining EFT and OCT. Besides, they have the same time complexity $\mathcal{O}(n^2m)$.

Greedy-Ant, a metaheuristic method, introduces heuristic searching into task prioritizing and a simple greedy strategy into machine allocating. Moreover, a global heuristic information η_{ij} considering forward and backward dependencies is defined to control the convergence of *Greedy-Ant*. Therefore, *Greedy-Ant* generates the better scheduling results than the list-based heuristic methods HEFT and PEFT. However, the time complexity of *Greedy-Ant* is higher than HEFT and PEFT.

The ACO and SA methods compared in the experiments fix an initial task sequence generated randomly. The solution space of machine allocating is limited and the performances rely on the quality of the initial task sequence.

V. CONCLUSION

This paper reconsiders ant colony system and presents a new scheme for ACS-based workflow scheduling. A new heuristic information based on forward and backward dependency is proposed to build transition probability for ants to generate task priorities. Simultaneously, a greedy minimum strategy for machine allocation is incorporated to complete scheduling. Experiments on both synthetic graphs and real application graphs demonstrate the effectiveness of *Greedy-Ant*. In the future, based on *Greedy-Ant*, we intend to consider some resources restricted cases under specific cloud environments, and formulate the problem with more QoS constraints. Further, we intend to improve *Greedy-Ant* in a hybrid heuristic manner.

REFERENCES

[1] M. R. Gary and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman, 1990.
[2] M. A. Tawfeek, A. El-Sisi, A. E. Keshk, and F. A. Torkey, "Cloud task scheduling based on ant colony optimization," in *Proc. 8th Int. Conf. Comput. Eng. Syst. (ICCES)*, Nov. 2013, pp. 64–69.
[3] W. N. Chen and J. Zhang, "An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 39, no. 1, pp. 29–43, Jan. 2009.
[4] S. Kianpisheh, N. M. Charkari, and M. Kargahi, "Ant colony based constrained workflow scheduling for heterogeneous computing systems," *Cluster Comput.*, vol. 19, no. 3, pp. 1053–1070, Sep. 2016.
[5] X. Liu and J. Liu, "A task scheduling based on simulated annealing algorithm in cloud computing," *Int. J. Hybrid Inf. Technol.*, vol. 9, no. 6, pp. 403–412, 2016.
[6] S. Chai, Y. Li, J. Wang, and C. Wu, "A list simulated annealing algorithm for task scheduling on network-on-chip," *J. Comput.*, vol. 9, no. 1, pp. 176–182, 2014.
[7] M. Masdari, F. Salehi, M. Jalali, and M. Bidaki, "A survey of PSO-based scheduling algorithms in cloud computing," *J. Netw. Syst. Manage.*, vol. 25, no. 1, pp. 122–158, Jan. 2016.
[8] Y. Xu, K. Li, J. Hu, and K. Li, "A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues," *Inf. Sci.*, vol. 270, pp. 255–287, Jun. 2014.
[9] S.-C. Chu and P.-W. Tsai, "Computational intelligence based on the behavior of cats," *Int. J. Innov. Comput., Inf. Control*, vol. 3, no. 1, pp. 163–173, 2007.
[10] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.
[11] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 3, pp. 682–694, Mar. 2014.
[12] M. Mollajafari and H. S. Shahhoseini, "An efficient ACO-based algorithm for scheduling tasks onto dynamically reconfigurable hardware using TSP-likened construction graph," *Appl. Intell.*, vol. 45, no. 3, pp. 695–712, Oct. 2016.
[13] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 53–66, Apr. 1997.
[14] F. Suter and S. Hunold. *Daggen: A Synthetic Task Graph Generator*, accessed on Apr. 26, 2017. [Online]. Available: https://github.com/frs69wq/daggen
[15] P. Maechling, E. Deelman, L. Zhao, R. Graves, G. Mehta, and N. Gupta *et al.*, "SCEC CyberShake workflows—Automating probabilistic seismic hazard analysis calculations," in *Workflows for e-Science*. London, U.K.: Springer, 2007, pp. 143–163.
[16] J. Livny, H. Teonadi, M. Livny, and M. K. Waldor, "High-throughput, kingdom-wide prediction and annotation of bacterial non-coding RNAs," *PLoS ONE*, vol. 3, no. 9, p. e3197, 2008.

BIN XIANG received the B.S. degree in electronic engineering from Southwest Jiaotong University, Chengdu, China, in 2013. He is currently pursuing the Ph.D. degree in information and communication engineering from the Beijing University of Posts and Telecommunications, Beijing, China. His research interests focus on mobile cloud computing, parallel and distributed computing, resource management, data analysis, and visual computing.

BIBO ZHANG received the B.S. degree in electronic engineering from the Beijing University of Posts and Telecommunications, Beijing, China, in 2015, where she is currently pursuing the master's degree in information and communication engineering. Her main research interests include mobile cloud computing, parallel and distributed computing, resource management, and data analysis.

LIN ZHANG received the B.S. and Ph.D. degrees from the Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 1996 and 2001, respectively. He was a Post-Doctoral Researcher with the Information and Communications University, Daejeon, South Korea, from 2000 to 2002. He is currently the Dean of School of Information and Communication Engineering, BUPT. He went to Singapore and held a research Fellow position with Nanyang Technological University, Singapore, from 2003 to 2004. He joined BUPT in 2004 as a Lecturer, then an Associate Professor in 2005, and a Professor in 2011. He served the university as the Director of Faculty Development Center and the Deputy Dean of Graduate School. He has authored more than 120 papers in referenced journals and international conferences. His research interests include mobile cloud computing and Internet of Things.