Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 6631752, 13 pages
https://doi.org/10.1155/2021/6631752

Research Article

Multijob Associated Task Scheduling for Cloud Computing Based on Task Duplication and Insertion

Lei Shi, Jing Xu, Lunfei Wang, Jie Chen, Zhifeng Jin, Tao Ouyang, Juan Xu, and Yuqi Fan

School of Computer Science and Information Engineering, Intelligent Interconnected Systems Laboratory of Anhui Province, Hefei University of Technology, Hefei, Anhui 230601, China

Correspondence should be addressed to Yuqi Fan; yuqi.fan@hfut.edu.cn

Received 27 November 2020; Revised 8 March 2021; Accepted 10 April 2021; Published 28 April 2021

Academic Editor: Yan Huang

With the emergence and development of various computer technologies, many jobs processed in cloud computing systems consist of multiple associated tasks which follow the constraint of execution order. The task of each job can be assigned to different nodes for execution, and the relevant data are transmitted between nodes to complete the job processing. The computing or communication capabilities of each node may be different due to processor heterogeneity, and hence, a task scheduling algorithm is of great significance for job processing performance. An efficient task scheduling algorithm can make full use of resources and improve the performance of job processing. The performance of existing research on associated task scheduling for multiple jobs needs to be improved. Therefore, this paper studies the problem of multijob associated task scheduling with the goal of minimizing the jobs' makespan. This paper proposes a task Duplication and Insertion algorithm based on List Scheduling (DILS) which incorporates dynamic finish time prediction, task replication, and task insertion. The algorithm dynamically schedules tasks by predicting the completion time of tasks according to the scheduling of previously scheduled tasks, replicates tasks on different nodes, reduces transmission time, and inserts tasks into idle time slots to speed up task execution. Experimental results demonstrate that our algorithm can effectively reduce the jobs' makespan.

1. Introduction

Due to the rapid development of cloud computing and cloud infrastructure, an increasing number of applications are migrating to the cloud. Because of the strong extensibility and need for cloud computing, many existing tasks need powerful computing power and computing resources. The core of cloud computing is to coordinate many computer resources together, such that users can obtain unlimited resources and process users' jobs. A cloud platform is built on a variety of computing and network components to achieve the supply and distribution of various resources, and different components have great heterogeneity. Usually, users submit jobs to the cloud platform. The cloud platform divides jobs into multiple associated tasks and assigns them to different components for execution. Components need to communicate to exchange data and get the final result.

The associated tasks of a job can usually be represented by a directed acyclic graph (DAG). Each node in the DAG represents a task, and the directed arc represents the sequence constraints between tasks [1, 2]. Only when the current node has completed can the tasks of the subsequent nodes be executed. The execution of the subsequent node may need the result of the predecessor node as the input, so there is data transmission between the nodes. Based on the need for job processing performance, there has been extensive research on associated task scheduling. The DAG-based task scheduling is widely used in DNA detection, image recognition, geological survey, climate prediction, and other practical applications [3]. A heuristic algorithm based on genetic algorithms and task duplication was proposed in [4]. A task duplication-based scheduling algorithm was introduced in [5]. The joint problem of task assignment and scheduling considering multidimensional task diversity

was modeled as a reverse auction with task owners being auctioneers; four auction schemes were designed to satisfy different application requirements [6, 7]. The preference of participants for different perceptual tasks is also a key factor to be considered in the auction mechanism, because assigning the least favorite task will hinder participants from participating in future perceptual tasks. A concept of "mutual preference degree" was proposed to capture a participant's preference, and a preference-based auction mechanism (PreAm) was designed to simultaneously guarantee individual rationality, budget feasibility, preference truthfulness, and price truthfulness [8]. A solution was proposed to detect and remove both short-term and long-term traffic redundancy through a two-layer redundancy elimination design [9]. A list-based scheduling algorithm called Predict Earliest Finish Time (PEFT) was proposed to introduce an optimistic cost table (OCT), which was used for task ranking and processor selection [10].

The processor may have idle time slots during processing multiple jobs. If the idle time slots can be fully utilized, the processing time of jobs can be reduced and the performance of job processing can be enhanced. A simple task scheduling algorithm may not be able to make full use of idle resources, so a more refined task scheduling algorithm is needed. In this paper, we study the associated task scheduling algorithm to minimize the jobs' makespan when the components of the cloud computing platform are heterogeneous.

The main contributions of this paper are summarized as follows.

(1) Aiming at minimizing the jobs' makespan, we model the associated task scheduling problem

(2) We propose a list scheduling algorithm based on task replication and task insertion (DILS), which combines dynamic completion time prediction, task replication, and task insertion. According to the previous task scheduling situation, the algorithm dynamically predicts the remaining completion time of tasks, replicates tasks on different nodes, reduces transmission time, and inserts tasks into idle time slots to accelerate task execution

(3) Experimental results demonstrate that our algorithm can effectively improve the performance of job processing

The remainder of this paper is organized as follows. Section 2 summarizes the related work. Section 3 proposes the problem model of associated task scheduling. Section 4 describes the multijob task scheduling algorithm. Section 5 compares our algorithm with other algorithms. Finally, Section 6 summarizes the work of this paper.

2. Related Work

In cloud computing platforms, there are many cases of processing multijob tasks, and the scheduling problem of multijob tasks is important to improve the performance of job processing. Previous studies usually took the makespan and energy consumption as the optimization objectives. Related works can be divided into the following categories.

The first category of the research investigates the scheduling strategy based on task priority. For example, a task scheduling algorithm based on the multipriority queue genetic algorithm (MPQGA) for heterogeneous computing systems was proposed [11]. In this paper, the authors used a genetic algorithm to assign priority to each task and used the heuristic method of the earliest completion time to complete the processor assignment; the authors also designed crossover, mutation, and fitness functions suitable for DAG scheduling. An algorithm was proposed to calculate the priority of each task to schedule tasks. The algorithm first processes the tasks with higher priority to satisfy the deadline [12]. A performance-effective task scheduling (PETS) algorithm was proposed to calculate the priority of each task based on task communication cost and average computation cost and then select the processor with the minimum earliest finishing time for each task [13].

The second category of the research studies the scheduling strategy based on a genetic algorithm. A learner genetic algorithm (denoted by LAGA) was presented to address static scheduling for processors in homogeneous computing systems [14]; the authors proposed two learning criteria named the steepest ascent learning criterion and next ascent learning criterion where they use the concepts of penalty and reward for learning; the algorithm exploits an efficient search method for solving a scheduling problem, such that the speed of finding a schedule could be accelerated and the trapping in local optimal could be alleviated; the algorithm also takes into consideration the idle time reuse criterion during the scheduling process to reduce the makespan. A decentralized scheduling algorithm based on genetic algorithms for the problem of DAG scheduling was proposed [15]; the genetic algorithm presents an effective method for optimization and could consider multiple criteria in the optimization process.

The third category of the research studies the scheduling strategy based on load balancing. A task scheduling mechanism was proposed based on two levels of load balance, which satisfies the dynamic task requirements of users and improves the utilization of resources [16]. A stochastic load balancing scheme was designed to provide a probabilistic guarantee against the resource overloading with virtual machine migration, while minimizing the total migration overhead [17].

The fourth category of the research focuses on QoS or heterogeneity during scheduling. A complex scene was considered where multiple moving MDs share multiple heterogeneous MEC servers, and a problem named the minimum energy consumption problem in a deadline-aware MEC system was formulated [18]. A network model aimed at improving user experience by pushing the scheduling problem to the task layer was proposed [19]. A QoE requirement was designed to generalize the QoS requirements of a task, which is the ratio requirement. Following this design, a corresponding scheduling policy was proposed to capture QoE for each task and then reached an application-aware transmission allocation. The q-coverage MLAS problem was investigated, which can guarantee that the aggregated nodes are

distributed evenly [20]; two algorithms were proposed by scheduling the communication tasks in the bottom-up and top-down manner, respectively; three algorithms were then proposed to ensure that the aggregation nodes are evenly distributed in the network with low delay; additionally, the method to extend the proposed algorithms for the BF-WSNs with multiple channels was also studied. Three allocation-aware task scheduling algorithms were proposed for a multicloud environment [21]; the algorithms are based on the traditional Min–Min and Max–Min algorithms and extended for multicloud environment; all the algorithms undergo three common phases, namely, matching, allocating, and scheduling, to fit them in the multicloud environment. A heuristic algorithm was presented for constructing streaming DAG which converts a large graph together to a DAG with the graph traversal algorithm, and then, a three-stage heterogeneous-aware cluster-scheduling algorithm was proposed to schedule the DAG to a heterogeneous cloud for parallel processing [22]; in the first stage, a parallel linear clustering algorithm was designed to cluster the DAG into a series of linear clusters with different granularities; in the second stage, a heterogeneous-aware load balancing algorithm was designed to map these clusters to different computing nodes in the cloud; in the last stage, a task sorting algorithm was designed to allocate the start time of these clusters as early as possible.

The fifth category of the research investigates the scheduling strategy based on task insertion. A Heterogeneous Earliest Finish Time (HEFT) algorithm was proposed to select the task with the highest upward rank value each time, then assign the task to the processor, and make use of task insertion to speed up the completion of the task and minimize the earliest completion time of the task [23]. A heterogeneous scheduling algorithm was proposed [24]; the algorithm takes the improved quantity as the calculation weight and communication weight and adopts an input task repeated selection strategy and an optimization strategy of idle time slot task insertion to improve the efficiency of task scheduling.

However, the existing works do not consider the scheduling problem in the case of multijob associated tasks. In this paper, we use DAGs to represent multijob associated tasks and then propose a task Duplication and Insertion algorithm based on List Scheduling (DILS), which can accelerate the execution of associated tasks to minimize the jobs' makespan.

3. Associated Task Scheduling Model

Example 1. An example of an associated task scheduling problem is shown in Figure 1. The job consists of four tasks, task set $T_k = \{t_1^k, t_2^k, t_3^k, t_4^k\}$ and directed arc set $\{e_{i,j}^k \mid t_i^k, t_j^k \in T_k\}$. The weight $c(t_i^k, t_j^k)$ of directed arc $e_{i,j}^k$ is the time required for data transmission from the server where task t_i^k is running to the server where task t_j^k will run. The weight of the link from task to server is the execution time required for the task to run on the server.

Assume that the job set with different priorities to be processed is $J = \{J_1, \cdots, J_k, \cdots\}$, where J_k is the kth DAG-based job with multiple associated tasks. The tuple $J_k = <T_k, E_k>$ represents the job J_k, where T_k represents the task set of J_k and E_k is the directed arc set $\{e_{i,j}^k \mid t_i^k, t_j^k \in T_k\}$ between tasks. Each directed arc represents the execution order constraint of two tasks. For example, $e_{i,j}^k$ means that the task t_j^k cannot be executed until the task t_i^k is executed; that is, t_i^k is the preceding task of t_j^k, t_j^k is the successive task of t_i^k, and t_i^k and t_j^k are the ith and jth tasks in job J_k, respectively. The directed arc also indicates that the execution of the subsequent task needs the result of the antecedent task execution. The weight $c(t_i^k, t_j^k)$ of directed arc $e_{i,j}^k$ is the time required for data to be transferred from the server where task t_i^k is running to the server running t_j^k. When two tasks connected by a directed arc run on the same server, weight $c(t_i^k, t_j^k) = 0$; that is, the communication within the same server is negligible.

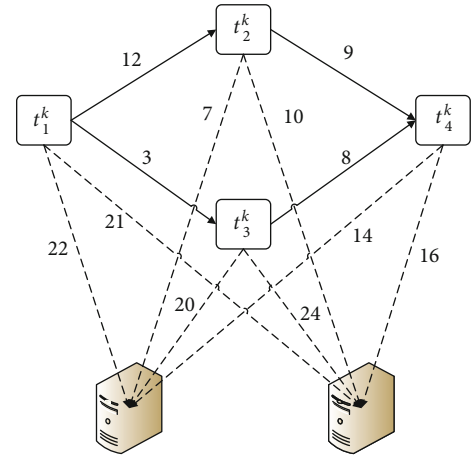


Figure 1: Architecture of the associated tasks and processor.

A DAG-based job is shown in Figure 2. In the DAG, task set $T_k = \{t_1^k, \cdots, t_i^k, \cdots, t_{10}^k\}$ contains 10 tasks and directed arc set $\{e_{i,j}^k \mid t_i^k, t_j^k \in T_k\}$ includes 15 directed arcs. For example, the directed arc from node t_2^k to node t_8^k indicates that t_2^k is the preceding task of task t_8^k, and the weight of this directed arc represents the communication time between the server running task t_2^k and the server running task t_8^k when two tasks are running on different servers.

Due to the heterogeneity of servers, the execution time of the same task may be different on different servers. $P = \{p_1, \cdots, p_s \cdots\}$ represents the set of servers that will run the tasks. We use an execution time matrix $W_k = T_k \times P$ to list all possible mapping between the task t_i^k and the server p_s. Each element $w(t_i^k, p_s)$ in the matrix W_k represents the time required for the task t_i^k to run on the server p_s. The execution time matrix W_k of DAG is shown in Table 1. The server set $P = \{p_1, p_2, p_3\}$ contains 3 servers, and the element of mapping task t_2^k and server p_2 indicates that the execution time of task t_2^k on server p_2 is 18.

Definition 2. In DAG-based job J_k, ingress task t_{in}^k is the task without any preceding tasks, and egress task t_{out}^k is the task without any successive tasks.

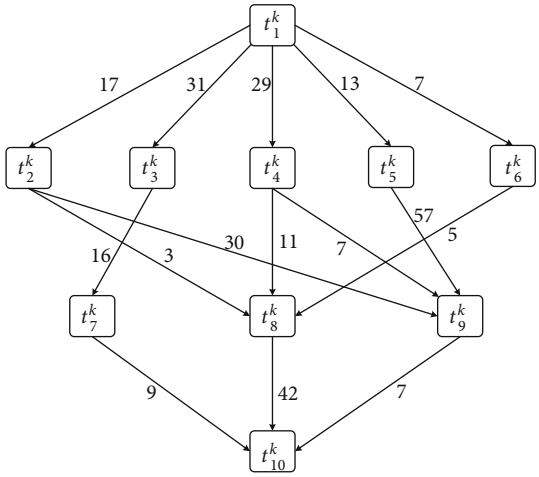


Figure 2: DAG-based job example.

Table 1: Execution time matrix example.

	p_1	p_2	p_3
t_1^k	22	21	36
t_2^k	22	18	18
t_3^k	32	27	43
t_4^k	7	10	4
t_5^k	29	27	35
t_6^k	26	17	24
t_7^k	14	25	30
t_8^k	29	23	36
t_9^k	15	21	8
t_{10}^k	13	16	33

For the DAG-based job shown in Figure 2, the ingress task and egress task are tasks t_1^k and t_{10}^k, respectively. If a job includes more than one ingress task, we add a virtual task node with zero computation cost as the virtual ingress task and add a directed arc from the virtual ingress task to each of the original ingress task nodes with zero weight.

Definition 3. $\mathrm{EST}(t_i^k, p_s)$,the earliest start time (EST) that task t_i^k can be executed on server p_s, is defined via Equation (1), where $\mathrm{EAT}(p_s)$ is the earliest time that server p_s is available, $\mathrm{pre}(t_i^k)$ is the set of the preceding tasks of task t_i^k, and $\mathrm{AFT}(t_j^k)$ is the actual finish time of task t_j^k:

$$\mathrm{EST}\left(t_i^k, p_s\right) = \max\left\{\mathrm{EAT}(p_s), \max_{t_j^k \in \mathrm{pre}\left(t_i^k\right)}\left\{\mathrm{AFT}\left(t_j^k\right) + c\left(t_j^k, t_i^k\right)\right\}\right\}, \quad \forall t_i^k \in T_k, \forall p_s \in P. \tag{1}$$

Definition 4. $\mathrm{EFT}(t_i^k, p_s)$,the earliest finish time (EFT) that server p_s can complete the execution of task t_i^k, is calculated via

$$\mathrm{EFT}\left(t_i^k, p_s\right) = \mathrm{EST}\left(t_i^k, p_s\right) + w\left(t_i^k, p_s\right), \quad \forall t_i^k \in T, \forall p_s \in P. \tag{2}$$

Definition 5. For job set J in which the jobs have different priorities, the makespan of job set J, $\Gamma(J)$, is the completion time of all the jobs and can be calculated via

$$\Gamma(J) = \max_{J_k \in J} \mathrm{AFT}\left(t_{\mathrm{out}}^k\right). \tag{3}$$

Given job set J, the DAG of each job, server set P, and the execution time of each job on each server, we need to schedule the jobs in J on the servers in P, so as to minimize $\Gamma(J)$, i.e., the makespan of job set J. The detailed notations are listed in Table 2.

4. Multijob Task Scheduling Algorithm

The single-job associated task scheduling problem with the aim of minimizing the makespan is an NP-hard problem [25]. Therefore, the multijob associated task scheduling problem is also an NP-hard problem [26].

In this section, a task Duplication and Insertion based on List Scheduling (DILS) algorithm is proposed, which incorporates dynamic finish time prediction, task replication, and task insertion. The algorithm dynamically predicts the remaining execution time of each task according to the scheduling of scheduled tasks. Then, the algorithm schedules the tasks according to the latest time remaining on the server, which can minimize the remaining time. After each task is scheduled, the algorithm can advance the start time of the task by adopting task replication and task insertion. The remaining execution time of each reserved task is updated when the task is scheduled. The algorithm DILS shown in Algorithm 1 consists of five parts: calculation of task remaining time, selection of tasks to be scheduled, server allocation of tasks to be scheduled, task replication, and task insertion.

4.1. Task Remaining Time Calculation. There is a weight for each task which is the total communication and computation time of all the subsequent tasks. A predicted remaining time (PRT) table maintains the weights of all the tasks, and each element $\mathrm{PRT}(t_i^k, p_s)$ in the table PRT represents the predicted remaining time required for executing all the successive tasks of task t_i^k $(1 \le i \le N)$, when it is allocated to server $p_s \in P$. The weight of task t_i^k is closely related to its successive tasks and the number of servers, and we let

$$A_{i,s}^k = \max_{t_j^k \in \mathrm{suc}\left(t_i^k\right)}\left\{\mathrm{MIN}_{p_t \in p}\left\{\mathrm{PRT}\left(t_j^k, p_t\right) + w\left(t_j^k, p_t\right) + c\left(t_i^k, t_j^k\right)\right\}\right\}, \tag{4}$$

Table 2: Table of notations.

Notation	Description
J_k	The kth DAG-based job with multiple associated tasks.
T_k	The task set of J_k.
E_k	The directed arc set $\left\{e_{i,j}^k \mid t_i^k, t_j^k \in T_k\right\}$ between tasks.
P	The set of servers.
t_i^k	The ith task of J_k.
$e_{i,j}^k$	The directed arc of t_i^k and t_j^k.
$c(t_i^k, t_j^k)$	The transmission time required for data.
p_s	The server that will run the tasks.
$w\left(t_i^k, p_s\right)$	The time required for the task t_i^k to run on the server p_s.
t_{in}^k	The task without any preceding tasks.
t_{out}^k	The task without any successive tasks.
$\text{pre}\left(t_i^k\right)$	The set of the preceding tasks of task t_i^k.
$\text{suc}\left(t_i^k\right)$	The set of the successive tasks of task t_i^k.
M	The number of servers.

Input: Server set P, job set J with each job's DAG, and execution time matrices
Output: Makespan of the jobs
for each job $J_K \in J$ **do**
  Calculate the predicted remaining time of each task via Eq. (6) and create the Prediction of Remaining Time (PRT) table;
  Create an empty ready task list and add the ingress task to the list;
  while ready task list is not empty **do**
    for each task t_i^k in ready task list **do**
      Compute the average path length of task t_i^k via Eq. (8);
    end for
    Select the task with the maximum average path length with Eq. (9);
    Assign the server leading to the minimum estimated path length via Eq.(10) to the task;
    Task duplication and task insertion;
    Update the ready task list;
  end while
end for
return the completion time of the last scheduled task.

Algorithm 1: Algorithm DILS.

where $\text{suc}(t_i^k)$ is the set of successive tasks of task t_i^k, and let

$$B_i^k = \frac{\sum_{t_j^k \in \text{suc}\left(t_i^k\right)} \sum_{p_t \in P} \text{PRT}\left(t_j^k, p_t\right)/M}{M}, \tag{5}$$

where M is the number of servers. Note that no matter which server runs the egress task, the weight of the exit task of each job is 0. That is, for any $p_t \in P$, $\text{PRT}(t_{\text{out}}^k, p_t) = 0$:

$$\text{PRT}\left(t_i^k, p_s\right) = \max\left\{A_{i,s}^k, B_i^k\right\}, \quad \forall t_i^k \in T, \forall p_s \in P. \tag{6}$$

The weight of task t_i^k is calculated via Equation (6). In each DAG, starting from the egress task to the ingress task, algorithm DILS recursively calculates backwards the weight of each task on each server and obtains the PRT table.

4.2. Selection of the Task to Be Scheduled. When all the preceding tasks of task t_i^k are scheduled, we call the task t_i^k *ready*. We create a ready task list (RTL) to maintain all ready tasks. Initially, the ready task list contains only one node, ingress task t_{in}^k, because only the ingress task of each DAG-based job is ready. We calculate the earliest start

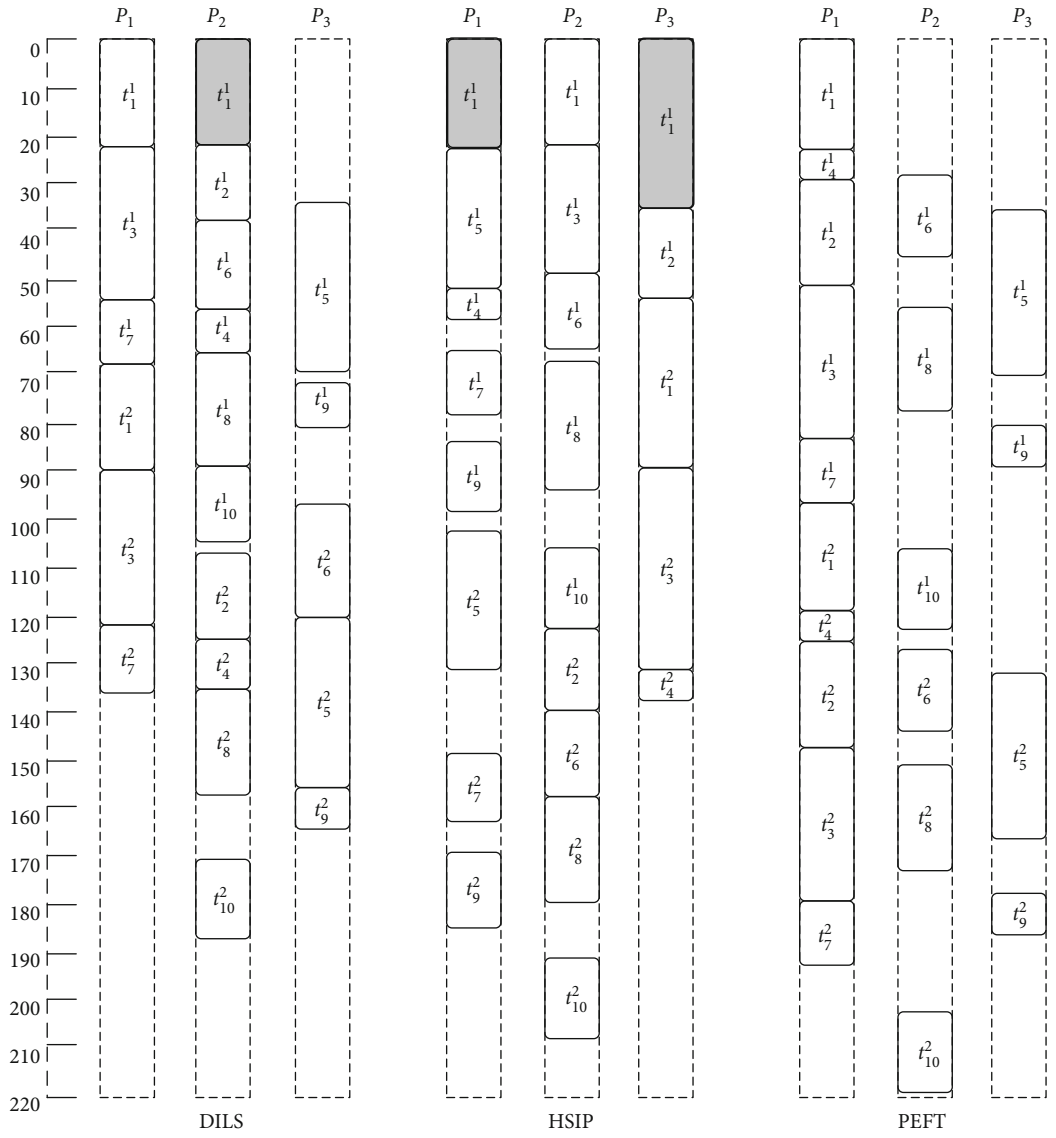


Figure 3: The example of scheduling result.

time (EST) of each task in the ready task list. The EST of the task t_i^k on the server p_s, t_i^k on server p_s, $\mathrm{EST}(t_i^k, p_s)$ is calculated via Equation (1). The estimated path length (EPL) of the task t_i^k assigned to the server p_s is calculated via

$$\mathrm{EPL}\left(t_i^k, p_s\right) = \mathrm{EST}\left(t_i^k, p_s\right) + w\left(t_i^k, p_s\right) + \mathrm{PRT}\left(t_i^k, p_s\right), \quad \forall t_i^k \in T_k, \forall p_s \in P. \tag{7}$$

Since each task may be assigned to any server $p_s \in P$, the average path length (APL) $\mathrm{APL}(t_i^k)$ of task t_i^k which is ready can be calculated with

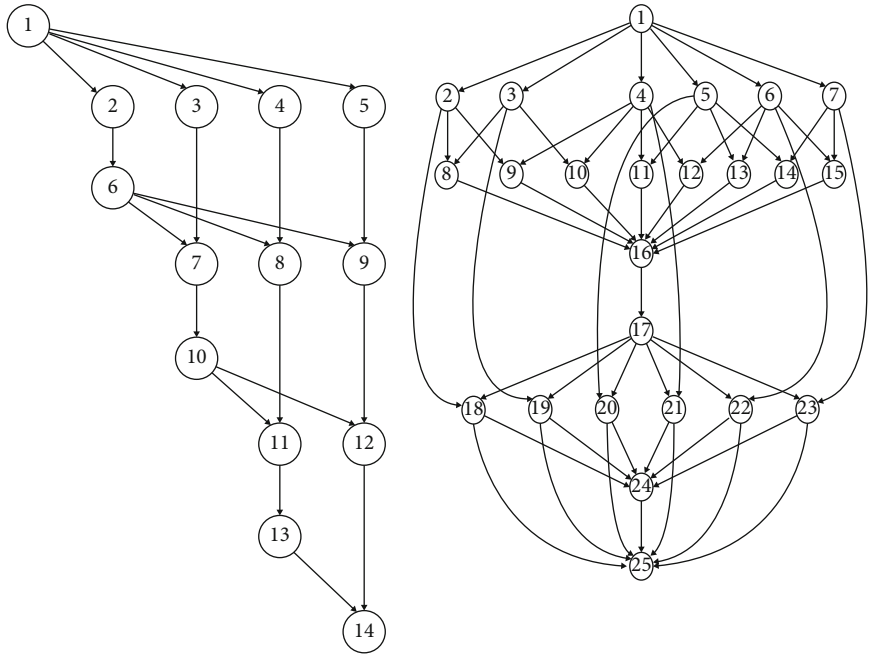
$$\mathrm{APL}\left(t_i^k\right) = \frac{\sum_{p_s \in P} \mathrm{EPL}\left(t_i^k, p_s\right)}{M}, \quad \forall t_i^k \in T_k. \tag{8}$$

Considering the difference in execution time of tasks on different paths, the task t_i^k which is ready and has the maximum average path length is selected as the task to be scheduled; that is,

$$t_i^k = \mathrm{argmax}_{t_j^k \in \mathrm{RTL}} \left\{ \mathrm{APL}\left(t_j^k\right) \right\}. \tag{9}$$

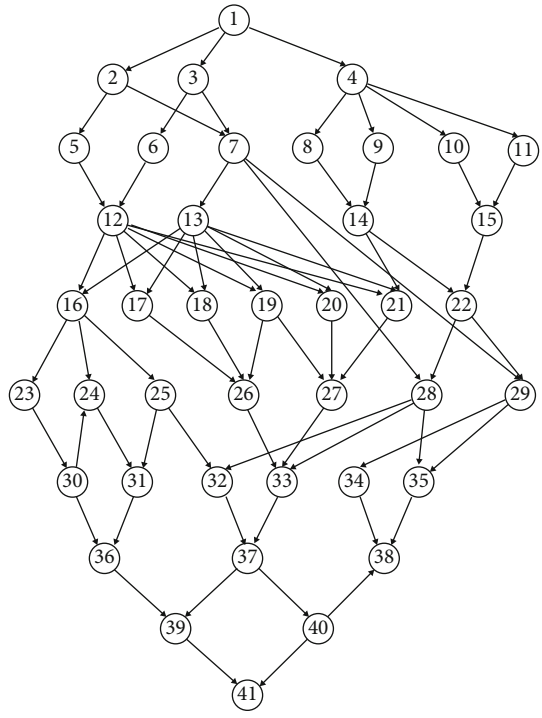
The selection of tasks is related to the EST of the task, and EST changes dynamically with the scheduling results of the previous tasks. Therefore, in the process of scheduling associated tasks, the selection of tasks to be scheduled will be changed dynamically.

4.3. Allocation of the Server for the Task to Be Scheduled. When the task t_i^k can get the minimum estimated path



(a) Gaussian Elimination

(b) Montage Workflow



(c) Molecular Dynamics

Figure 4: Real-world DAGs.

length on the server p_s, we assign the to-be-scheduled task t_i^k to server p_s to reduce the makespan; that is,

$$p_s = \text{argmin}_{p_t \in P}\left\{\text{EPL}\left(t_i^k, p_t\right)\right\}. \tag{10}$$

For task t_i^k, if multiple servers can get the same minimum estimated path length, we randomly assign task t_i^k to one of the servers. The actual start time of the task t_i^k can be calculated by Equation (2). When the server of the task to be scheduled is assigned, some other tasks may be ready to be scheduled, so the ready task list needs to be updated.

4.4. Task Duplication. Task duplication obtains multiple copies of the task t_i^k and then assigns the task copies to different servers for execution, such that the direct successive tasks of task t_i^k can

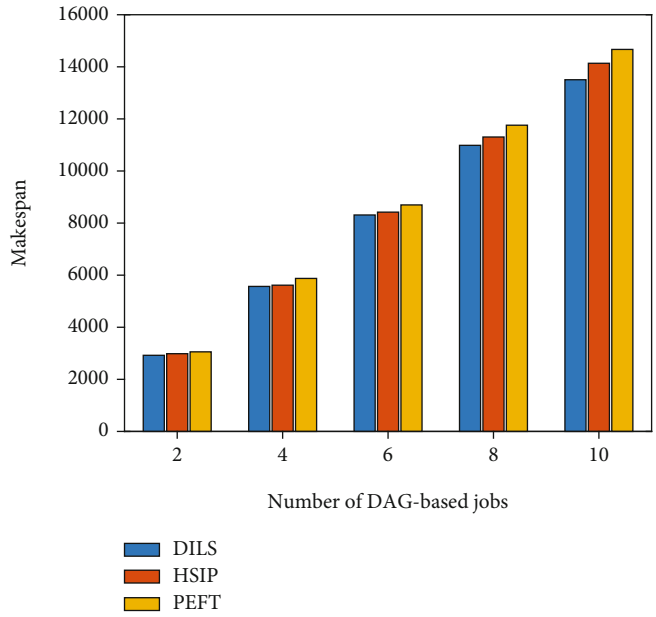


Figure 5: The makespan with the different numbers of DAG-based jobs.

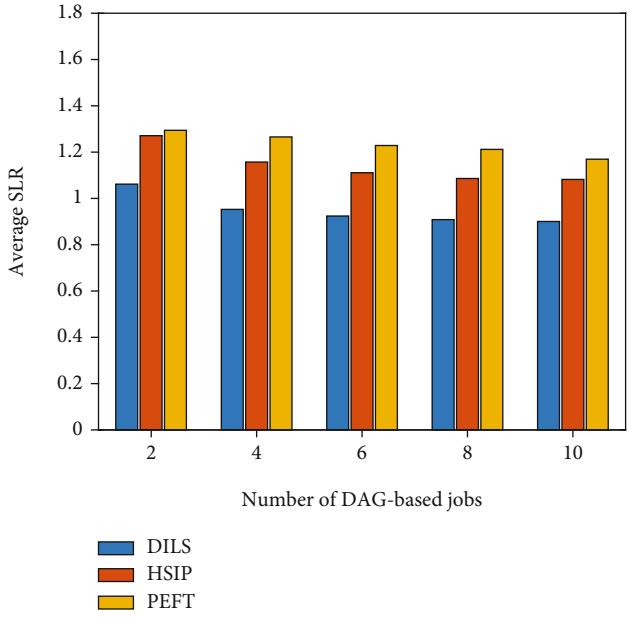


Figure 6: The average SLR with the different numbers of DAG-based jobs.

be executed immediately after the execution of t_i^k, instead of waiting for the data generated by task t_i^k to be transmitted through the network. Task duplication works as follows.

(1) Each task t_j^k in $\text{pre}(t_i^k)$ is assigned a time weight τ_j^k, that is, the time required for the data generated by the preceding task of task t_i^k to be transmitted to the processor p_s where task t_j^k is to be executed

(2) All the preceding tasks in $\text{pre}(t_i^k)$ are sorted in nonascending order according to the time weights

(3) Each of the preceding tasks in $\text{pre}(t_i^k)$ is iterative processed. Schedule a copy of task $t_j^k \in \text{pre}(t_i^k)$ to the earliest idle time slot (ITS) of server p_s, if the following conditions are met: (I) $\tau_j^k > \text{EAT}(p_s)$; that is, τ_j^k is greater than the earliest available time of processor p_s; (II) there is an idle time slot for the copy of t_j^k; and (III) task t_i^k can start earlier by duplicating t_j^k

4.5. Task Insertion. Task insertion inserts a task into an idle time slot on a processor which is occupied by some tasks after the time slot. Task insertion works as follows.

(1) When task t_i^k is to be scheduled on server p_s, we search all the idle time slots on server p_s. When the idle time slot meets the following conditions: (I) $\text{EST}(t_i^k, p_s)$, the earliest start time of task t_i^k, is no earlier than the start time of the idle time slot; (II) EFT (t_i^k, p_s), the earliest finish time of task t_i^k on processor p_s, is no later than the end time of the idle time slot; we select this idle time slot to insert task t_i^k

(2) If more than one idle time slot satisfies the above conditions, we choose the idle time slot with the smallest difference between the length of the idle time slot and the execution time of task t_i^k

4.6. Time Complexity. Suppose the number of DAGs is K, the number of processors is M, and the number of task nodes in each DAG is $N_1, N_2, \ldots, N_K$, respectively. For the ith $(1 \le i \le k)$ DAG, DILS requires the computation of a PRT table, which consumes $O(M \times N_i^2)$ time. The time complexity of calculating the earliest start time and estimated path length of the task on the processor is $O(M^2 \times N_i^2)$. By executing

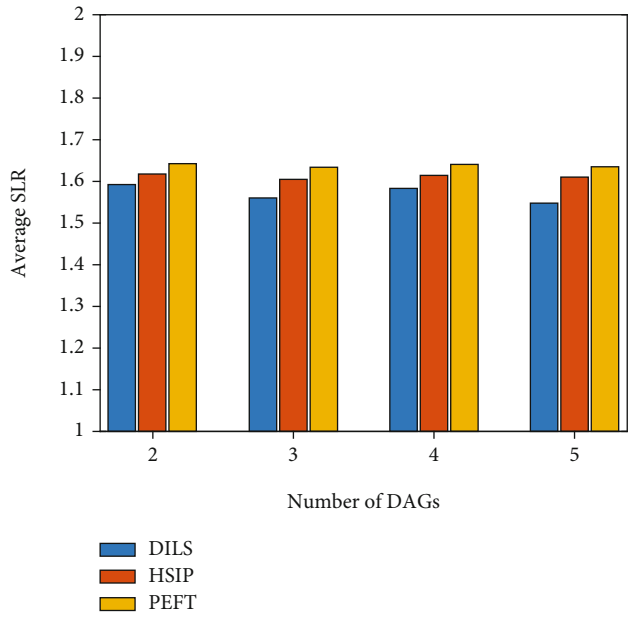


Figure 7: Average SLR with different numbers of DAG-based jobs for Gaussian Elimination.

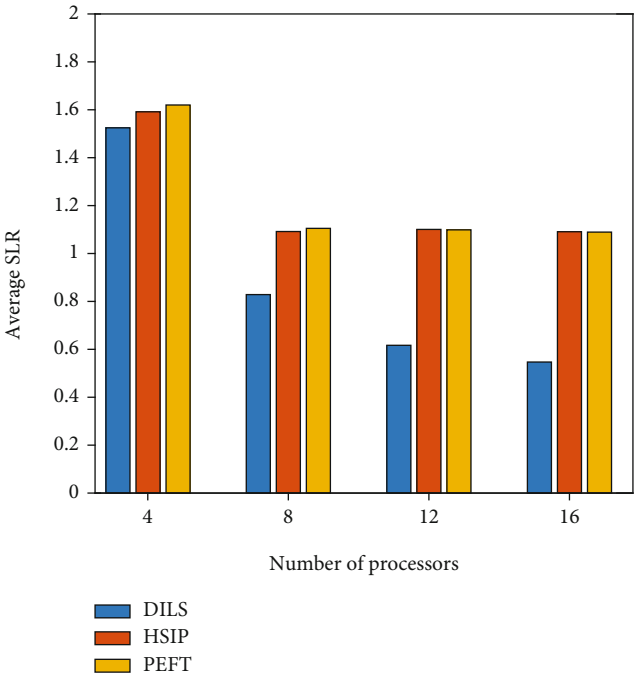


Figure 8: Average SLR with different numbers of processors for Gaussian Elimination.

the insertion mechanism in $O(M \times N_i^2)$ and executing task replication in $O(M \times N_i^2)$, the time complexity of updating the task ready list is $O(N_i^2)$. Therefore, the time complexity of Algorithm 1 is $O(M^2 \times N_1^2) + O(M^2 \times N_2^2) + \cdots + O(M^2 \times N_k^2) = O(M^2 \times \Sigma_i^k N_i^2)$. Algorithms HSIP [24] and PEFT [10] are also scheduling strategies based on task insertion, which

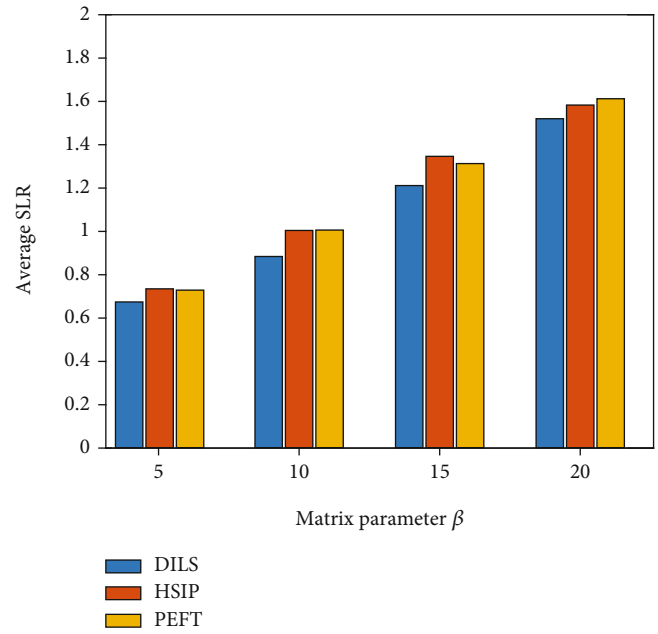


Figure 9: The average SLR with different matrix parameter β.

deal with the case of a single DAG with the time complexity of $O(N^2 \times M)$.

5. Simulation

In this section, we compare the performance of the proposed algorithm DILS with two state-of-the-art algorithms PEFT [10] and HSIP [24]. For this purpose, we consider the effect of the number of DAG-based jobs and processors on the performance of the algorithms. We also investigate the influence of some important parameters on the performance of the algorithms.

Example 7. The scheduling results of the three algorithms for job set $J_s = <J_1, J_2>$ are illustrated in Figure 3, where DAGs of jobs J_1 and J_2 are the same as that described in Figure 2 and Table 1. The gray shadowed block represents the task to be copied by algorithms DILS and HSIP. In this example, we can see that algorithm DILS has better results than HSIP and PEFT.

5.1. Simulation Setup. In this paper, we will evaluate the performance of the three algorithms in terms of the scheduling length ratio (SLR). SLR is the ratio of the scheduling length to the minimum scheduling length by ignoring the communication time, which can be calculated by

$$\mathrm{SLR} = \frac{\sum_{J_k \in J} \mathrm{makespan}}{\sum_{J_k \in J} \sum_{t_i^k \in \mathrm{CP}_{\mathrm{MIN}}} \min_{p_s \in P} \left\{ w\left(t_i^k, p_s\right) \right\}}, \tag{11}$$

where $\mathrm{CP}_{\mathrm{MIN}}$ is the minimum length of the critical path in the DAG-based job after ignoring the communication time between tasks, while the critical path of a DAG-based job is the longest path from the ingress task to the egress task.

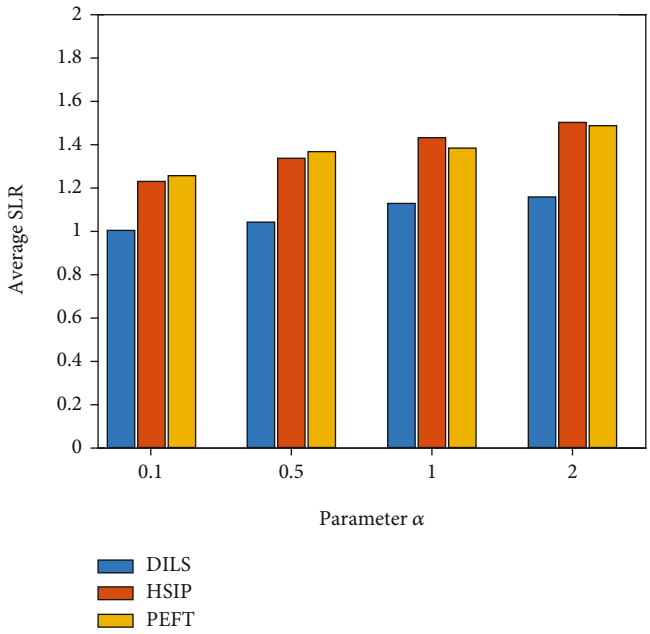


Figure 10: Average SLR with different α for Montage Workflow.

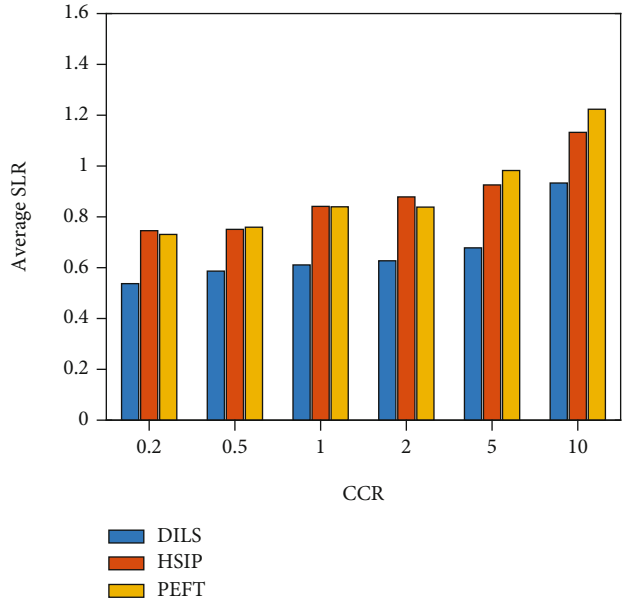


Figure 11: Average SLR with different CCR for Montage Workflow.

In the simulations, we use DAG-based jobs randomly generated by a DAG generator and real-world DAG-based jobs. The DAG generator uses the same parameters as in [25, 27]. The DAG topology parameters are as follows:

(1) DAG average calculation time $c_{\overline{\mathrm{DAG}}}$: indicating the average execution time of the tasks in the DAG, which is randomly set during the simulation

(2) Communication calculation ratio CCR: the ratio of the average communication time and the average execution time; the larger the value, the more communication-intensive the DAG-based job; the smaller the value, the more computation-intensive the DAG-based job

(3) Heterogeneous parameter α: representing the task execution time range on different processors; the larger the value, the more heterogeneous the processors

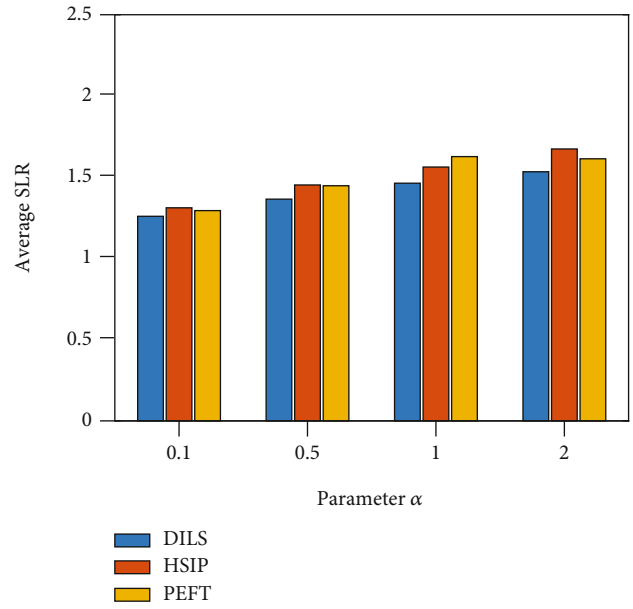


Figure 12: Average SLR with different α for Molecular Dynamics Code.

We select three classic DAG-based jobs from the real-world applications in the simulations.

(1) Gaussian Elimination: it is used in linear algebraic programming for solving linear equations as shown in Figure 4(a). The number of nodes is $N = (\beta^2 + \beta - 2)/2$ according to matrix parameter β

(2) Montage Workflow: it is applied to construct astronomical image mosaic. An example of Montage Workflow is shown in Figure 4(b)

(3) Molecular Dynamics Code: it is an algorithm to implement the atomic and the molecular physical motion. An example of the Molecular Dynamics Code is depicted in Figure 4(c)

5.2. Performance of Algorithm DILS. Figure 5 shows the effect of changing the number of DAG-based jobs on the performance of the makespan of the three algorithms. The number of servers is set to 4, DAG-based jobs are randomly generated by the DAG generator, and the parameters CCR and α are both set to 1. It can be seen from Figure 5 that the makespan increases with the increase of DAG-based jobs. In short, among the three algorithms, DILS achieves the best performance, followed by algorithm HSIP and algorithm PEFT.

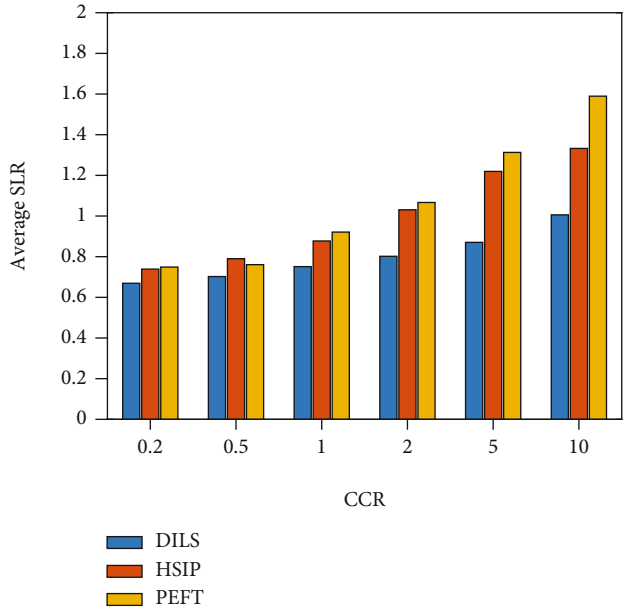


Figure 13: Average SLR with different CCR for Molecular Dynamics Code.

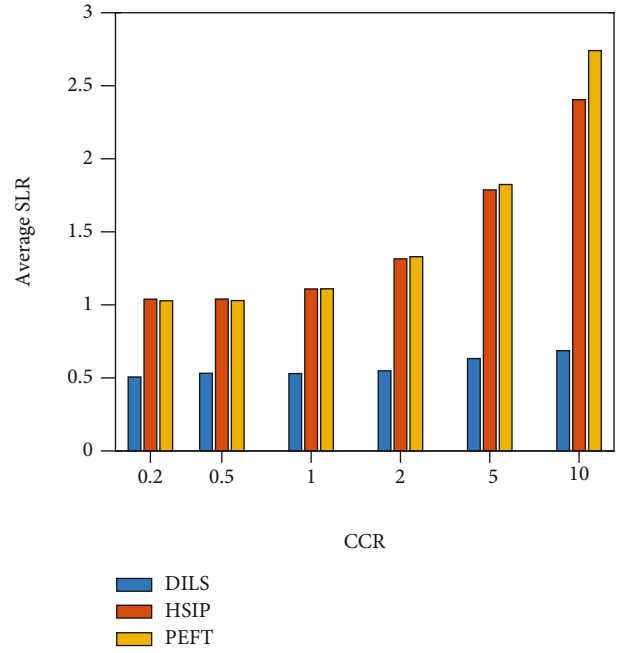


Figure 15: Average SLR with different CCR for Gaussian Elimination.

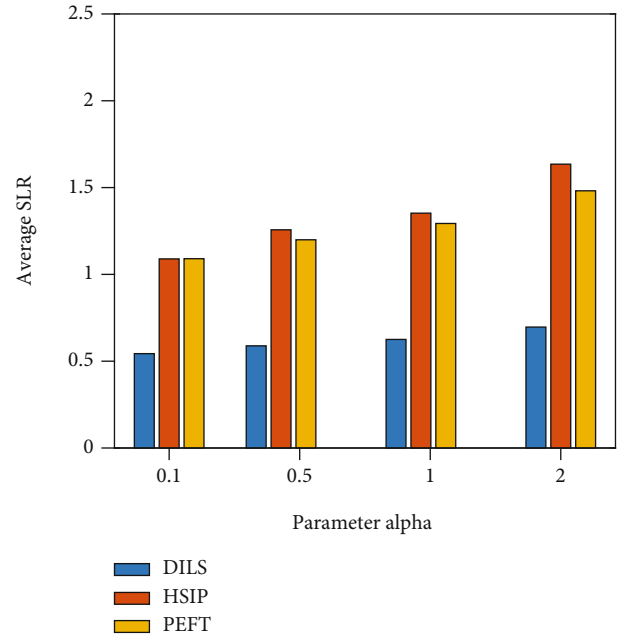


Figure 14: Average SLR with different α for Gaussian Elimination.

When there are 10 DAG-based jobs, the algorithm DILS improves the performance of algorithm HSIP and PEFT by 4.5% and 7.9%, respectively.

Figure 6 shows the effect of the number of DAG-based jobs on the three algorithms on the average SLR. The number of servers is set to 8, DAG-based jobs are randomly generated by the generator, and the parameters CCR and α are both set to 1. The average SLR generated by the three algorithms decreases with the increase of the number of DAG-based jobs, because more idle time slots are utilized. Among the three algorithms, DILS can get the minimum average SLR, because DILS makes full use of the idle time slots. When the number of DAG-based jobs increases from 2 to 10, the performance of algorithm DILS compared with algorithm HSIP and PEFT is improved from 16.3% to 17.6% and from 17.9% to 25.0%, respectively.

In Figure 7, we show the impact of the number of DAG-based jobs on the average SLR for Gaussian Elimination jobs. The number of processors is 4, the parameter CCR is set to 1, and α is set as 0.1. As the number of DAG-based jobs increases, the increase of average SLR is small. The difference between the three algorithms is obvious. However, our algorithm outperforms the other two algorithms.

Figure 8 illustrates the average SLR of the three algorithm changes with the increasing number of processors for Gaussian Elimination jobs. With the increase in the number of processors, the average SLR of our algorithm decreases significantly. Obviously, compared with other algorithms, our algorithm achieves better performance.

5.3. Impact of Parameters. Figure 9 illustrates the average SLR performance versus different matrix size parameter β for the Gaussian Elimination jobs when the number of servers is 2. It can be observed that the average SLR of the three algorithms increases with the increase of β. The jobs with a larger β consist of more tasks and hence require more computation and transfer time for all the associated tasks. The performance improvement of algorithm DILS on algorithms HSIP and PEFT is up to 11.9% and 12.1%, respectively.

Figure 10 describes the average SLR for Montage Workflow. Parameter α varies in $\{0.1, 0.5, 1, 2\}$, when the parameter CCR = 1 and the numbers of DAG-based jobs and

servers are set as 5 and 4, respectively. The performance of algorithm DILS compared with algorithms HSIP and PEFT is improved from 18.4% to 22.9% and from 20.1% to 22.4%, respectively. For Figure 11, the parameter CCR increases from 0.2 to 10, when $\alpha = 1$ and the numbers of DAG-based jobs and servers are set to 5 and 8, respectively. The average SLR increases with the increase of the parameter CCR, since the communication between tasks consumes more time as parameter CCR increases. It can be seen that the algorithm DILS always achieves the best performance among the three algorithms.

Figure 12 depicts the impact of parameter α on the average SLR for Molecular Dynamics Code. The parameter CCR is equal to 1, and the numbers of DAG-based jobs and servers are set to 5 and 4, respectively. It can be seen that the average SLR of the three algorithms increases with the increase of heterogeneous parameter α. Algorithm DILS always obtains the smallest results.

Figure 13 shows the impact of CCR on average SLR for Molecular Dynamics Code, when $\alpha = 1$ and the numbers of DAG-based jobs and servers are set to 5 and 8, respectively. The performance of algorithm DILS compared with HSIP and PEFT is improved from 9.4% to 24.5% and from 10.6% to 36.7%, respectively.

Figure 14 describes the average SLR for Gaussian Elimination with different parameter α varying in $\{0.1, 0.5, 1, 2\}$, when the parameter CCR = 1 and the numbers of DAG-based jobs and servers are set as 5 and 16, respectively. In general, algorithm DILS always achieves the best performance among the three algorithms, and algorithm HSIP performs better than algorithm PEFT. With the heterogeneous parameter α increasing from 0.1 to 2, the average SLR of algorithm DILS is improved from 50% to 57.3% and from 50.2% to 53% compared with HSIP and PEFT, respectively.

Figure 15 illustrates the average SLR for Gaussian Elimination with different parameter CCR increasing from 0.2 to 10, when $\alpha = 1$ and the numbers of DAG-based jobs and servers are set to 5 and 16, respectively. Compared with the other two algorithms, the average SLR performance of DILS increases from 51.3% to 71.5% and from 50.7% to 75%, respectively.

6. Conclusion

Careful multijob task scheduling is the key to achieve efficient job processing. This paper studied the problem of associated task scheduling of multiple jobs with the aim of minimizing jobs' makespan. We propose a task Duplication and Insertion algorithm based on List Scheduling (DILS). The algorithm combines dynamic prediction of task completion time, task replication, and task insertion. For multiple jobs, the expected completion time of their associated tasks is calculated to determine which task is scheduled and which processor to run the task. Some tasks are copied to different processors to reduce the transmission delay. There are many jobs to be processed, and hence, there may be idle time slots on the processor. Therefore, the tasks that meet the insertion conditions can be inserted into the idle time slots to speed up the execution of the jobs. Simulation results demonstrated that algorithm DILS achieved good performance.

Data Availability

There is no dataset for this article.

Disclosure

An earlier version of the paper has been presented as a conference paper in WASA2020: International Conference on Wireless Algorithms Systems and Applications.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This document is the result of the Key Research and Development Project in Anhui Province (Grant No. 201904a06020024), National Key Research and Development Plan (Grant No. 2018YFB2000505), and National Natural Science Foundation of China (Grant No. 61806067).

References

[1] W. Chen, G. Xie, R. Li, Y. Bai, C. Fan, and K. Li, "Efficient task scheduling for budget constrained parallel applications on heterogeneous cloud computing systems," *Future Generation Computer Systems*, vol. 74, no. C, pp. 1–11, 2017.

[2] H. Arabnejad and J. Barbosa, "Fairness resource sharing for dynamic workflow scheduling on heterogeneous systems," in *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, pp. 633–639, Leganes, Spain, 2012.

[3] S. K. Panda and P. K. Jana, "Efficient task scheduling algorithms for heterogeneous multi-cloud environment," *Journal of Supercomputing*, vol. 71, no. 4, pp. 1505–1533, 2015.

[4] T. Tsuchiya, T. Osada, and T. Kikuno, "A new heuristic algorithm based on GAs for multiprocessor scheduling with task duplication," in *Proceedings of 3rd International Conference on Algorithms and Architectures for Parallel Processing*, pp. 295–308, Melbourne, VIC, Australia, 1997.

[5] R. Bajaj and D. P. Agrawal, "Improving scheduling of tasks in a heterogeneous environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 2, pp. 107–118, 2004.

[6] Z. Duan, W. Li, and Z. Cai, "Distributed auctions for task assignment and scheduling in mobile crowdsensing systems," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 635–644, Atlanta, GA, USA, 2017.

[7] Z. Cai, Z. Duan, and W. Li, "Exploiting multi-dimensional task diversity in distributed auctions for mobile crowdsensing," *IEEE Transactions on Mobile Computing*, no. 99, p. 1, 2020.

[8] Z. Duan, W. Li, X. Zheng, and Z. Cai, "Mutual-preference driven truthful auction mechanism in mobile crowdsensing," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1233–1242, Dallas, TX, USA, 2019.

[9] L. Yu, H. Shen, K. Sapra, L. Ye, and Z. Cai, "CoRE: cooperative end-to-end traffic redundancy elimination for reducing cloud bandwidth cost," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 2, pp. 446–461, 2017.

[10] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 682–694, 2014.

[11] Y. Xu, K. Li, J. Hu, and K. Li, "A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues," *Information Sciences*, vol. 270, pp. 255–287, 2014.

[12] T. Choudhari, M. Moh, and T. Moh, "Prioritized task scheduling in fog computing," in *Proceedings of the ACMSE 2018 Conference*, pp. 1–8, Richmond, Kentucky, USA, 2018.

[13] E. Ilavarasan, P. Thambidurai, and R. Mahilmannan, "Performance effective task scheduling algorithm for heterogeneous computing system," in *4th International Symposium on Parallel and Distributed Computing (ISPDC'05)*, pp. 28–38, Lillie, France, 2005.

[14] H. Izadkhah, "Learning based genetic algorithm for task graph scheduling," *Applied Computational Intelligence and Soft Computing*, vol. 2019, 15 pages, 2019.

[15] F. Pop, C. Dobre, and V. Cristea, "Genetic algorithm for dag scheduling in grid environments," in *IEEE International Conference on Intelligent Computer Communication & Processing*, Cluj-Napoca, Romania, 2009.

[16] Y. Fang, F. Wang, and J. Ge, "A task scheduling algorithm based on load balancing in cloud computing," in *International conference on web information systems and mining*, pp. 271–277, Berlin, Heidelberg, 2010.

[17] L. Yu, L. Chen, Z. Cai, H. Shen, Y. Liang, and Y. Pan, "Stochastic load balancing for virtual resource management in datacenters," *IEEE Transactions on Cloud Computing*, vol. 8, no. 2, pp. 459–472, 2020.

[18] T. Zhu, T. Shi, J. Li, Z. Cai, and X. Zhou, "Task scheduling in deadline-aware mobile edge computing systems," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4854–4866, 2019.

[19] X. Zheng, Z. Cai, J. Li, and H. Gao, "A study on application-aware scheduling in wireless networks," *IEEE Transactions on Mobile Computing*, vol. 16, no. 7, pp. 1787–1801, 2017.

[20] Z. Cai and Q. Chen, "Latency-and-coverage aware data aggregation scheduling for multihop battery-free wireless networks," *IEEE Transactions on Wireless Communications*, vol. 20, no. 3, pp. 1770–1784, 2021.

[21] S. K. Panda, I. Gupta, and P. K. Jana, "Task scheduling algorithms for multi-cloud systems: allocation-aware approach," *Information Systems Frontiers*, vol. 21, no. 2, pp. 241–259, 2019.

[22] K. Hu, G. Zeng, S. Ding, and H. Jiang, "Cluster-scheduling big graph traversal task for parallel processing in heterogeneous cloud based on dag transformation," *IEEE Access*, vol. 7, pp. 77070–77082, 2019.

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

[24] G. Wang, Y. Wang, H. Liu, and H. Guo, "HSIP: a novel task scheduling algorithm for heterogeneous computing," *Scientific Programming*, vol. 2016, 11 pages, 2016.

[25] Y. Fan, L. Tao, and J. Chen, "Associated task scheduling based on dynamic finish time prediction for cloud computing," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, Dallas, Texas, USA, 2019.

[26] J. Ullman, "Np-complete scheduling problems," *Journal of Computer and System Sciences*, vol. 10, no. 3, pp. 384–393, 1975.

[27] D. Cordeiro, G. Mounie, P. Swann, D. Trystram, J.-M. Vincent, and F. Wagner, "Random graph generation for scheduling simulations," in *Proceedings of the 3rd International Icst Conference on Simulation Tools and Techniques (SIMUTools'10)*, Torremolinos, Malaga, Spain, March 15-19 2010.