Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 9529022, 16 pages
https://doi.org/10.1155/2021/9529022

*Research Article*

# List Scheduling Algorithm Based on Virtual Scheduling Length Table in Heterogeneous Computing System

**Naqin Zhou,[1] Xiaowen Liao,[2] Fufang Li,[2,3] Yuanyong Feng,[2] and Liangchen Liu[2]**

[1]*Cyberspace Institute of Advanced Technology, Guangzhou University, Guangzhou 510006, China*
[2]*School of Computer Science and Cyber Engineering, Guangzhou University, Guangzhou 510006, China*
[3]*Guangzhou Xuanyuan Research Institute Co. Ltd, Guangzhou 510006, China*

Correspondence should be addressed to Fufang Li; liff@gzhu.edu.cn

Received 3 October 2021; Accepted 20 November 2021; Published 11 December 2021

Academic Editor: Xiaojie Wang



Edge computing needs the close cooperation of cloud computing to better meet various needs. Therefore, ensuring the efficient implementation of applications in cloud computing is not only related to the development of cloud computing itself but also affects the promotion of edge computing. However, resource management and task scheduling strategy are important factors affecting the efficient implementation of applications. Therefore, aiming at the task scheduling problem in cloud computing environment, this paper proposes a new list scheduling algorithm, namely, based on a virtual scheduling length (BVSL) table algorithm. The algorithm first constructs the predicted remaining length table based on the prescheduling results, then constructs a virtual scheduling length table based on the predicted remaining length table, the current task execution cost, and the actual start time of the task, and calculates the task priority based on the virtual scheduling length table to make the overall path the longest task is scheduled first, thus effectively shorten the scheduling length. Finally, the processor is selected for the task based on the predicted remaining length table. The selected processor may not be the earliest for the current task, but it can shorten the finish time of the task in the next phase and reduce the scheduling length. To verify the effectiveness of the scheduling method, experiments were carried out from two aspects: randomly generated graphs and real-world application graphs. Experimental results show that the BVSL algorithm outperforms the latest Improved Predict Priority Task Scheduling (IPPTS) and RE-18 scheduling methods in terms of makespan, scheduling length ratio, speedup, and the number of occurrences of better quality of schedules while maintaining the same time complexity.

## 1. Introduction

Edge computing and cloud computing are widely used in various fields. Especially with the development of Internet of Things and 5G technology, a variety of access devices and mobile devices are growing explosively. Although cloud computing has strong performance, it is difficult to meet the real-time and sufficient bandwidth requirements in the face of a large number of data transmission and device connection requirements [1]. Edge computing plays an important role in the Internet of Things because it is close to the client, which can provide near-end low latency services and ensure the security and privacy of data. In recent years, many scholars have studied it from different angles to better promote edge computing, such as cloud edge collaborative computing offload [2–5], framework [6–9], mobile edge computing offload [10–16], and pervasive edge computing offload [17, 18]. A lot of work is also done to solve the trust and security [19–23] of data in the Internet of Things and data transmission [24] and collection [25] in wireless sensor networks. However, to better meet various needs, edge computing still needs the close cooperation of cloud computing. Therefore, how to schedule effectively in the cloud computing environment is not only related to the development of cloud computing itself but also affects the promotion of edge computing.

In cloud computing, most large applications do not rely on a single processor but use multiple processors for distributed and parallel computing. Generally, cloud computing resources are located in different geographical locations.

Each component in the environment has its operating system and is connected to the same network [26]. In essence, these resources are heterogeneous. They are connected through a high-speed network to form a heterogeneous system. Heterogeneous computing systems are highly competitive because they can provide parallel processing and high performance at a low cost [27], which is one of the reasons why they are widely used in scientific and industrial applications. More than half of the world's top ten supercomputing systems use GPU or CPU accelerator heterogeneous architectures, which are designed to maximize performance and efficiency [28, 29]. This also shows that heterogeneous computing systems will continue to be widely concerned and applied.

The efficiency improvement of heterogeneous computing systems is not only reflected in the optimization of hardware but also important for the effective utilization of internal computing resources. The application in a heterogeneous computing system can be decomposed into many subtasks with dependency and priority constraints. In this way, multiple processors can be used to execute tasks in parallel to minimize the scheduling length, which is usually represented in the form of a directed acyclic graph (DAG) [30]. Each node in the graph represents a task with different execution costs on different processors, and the weight on each edge also represents the communication cost between tasks. Therefore, how to schedule tasks in heterogeneous computing systems is a key factor to improve system performance. However, such problems have been proven to be NP-complete [31, 32]. To solve this problem, many heuristic-based algorithms [33–48] have been proposed in recent years, which are aimed at minimizing the scheduling length and other goals. These algorithms can be roughly divided into three categories: list scheduling, clustering scheduling, and duplication scheduling.

The list scheduling algorithm [33–38, 40, 41, 46] is relatively common. This type of algorithm has two main phases: task prioritization and processor selection. The first phase is mainly to sort the tasks according to the defined priority calculation function, and the latter phase is to select the tasks in the sorting list and choose an optimal processor to execute them until there are no executable tasks.

The clustering scheduling algorithm [43, 47–49] also has two phases. The first phase is mainly to analyze the characteristics of tasks or processors and cluster tasks or processors according to different clustering conditions. The second phase is scheduled based on the previous clustering. The main advantage of clustering is that it can reduce the cost of communication between tasks. However, for a heterogeneous system, the difference between task execution cost and intertask communication cost is large, and it will be difficult to select suitable clustering conditions. In reference [50], the authors compared many list scheduling and clustering scheduling algorithms and believe that simple and low complexity algorithms tend to have stronger competitiveness, while complex algorithms can only show advantages under certain conditions.

The idea of the duplication scheduling algorithm [39, 42, 44, 45, 51, 52] is to repeatedly schedule some tasks on different processors. The purpose of this is to reduce the delay in the execution of tasks due to communication time. Although the scheduling length can be shortened to a certain extent, the complexity of the algorithm is high and sacrifices a lot of processor resources.

List scheduling algorithm has lower complexity than clustering and duplication scheduling algorithm and can achieve good scheduling results, so task scheduling research also focuses more on list scheduling algorithms. However, these methods also have some shortcomings: the algorithm only analyzes the current task or its impact on subsequent tasks and lacks global considerations, which makes it difficult for the scheduling sequence generated in the priority phase to obtain better results. The calculation method of the priority is greatly affected by the system, and there are often errors of different degrees, which lead to poor stability.

To this end, we propose a new list scheduling algorithm, namely, based on a virtual scheduling length (BVSL) table algorithm to minimize the scheduling length of an application in heterogeneous computing systems. The experiment is based on randomly generated graphs and related real-world application graphs. The results show that the BVSL algorithm can obtain better scheduling results than the algorithms proposed in IPPTS [34] and literature [46]. The main contributions of the algorithm are as follows:

(1) To solve the priority ordering problem in list scheduling, a new concept of "virtual scheduling length table" is proposed. By using the virtual scheduling length table in the task prioritization phase, the priority of the task can be considered as a whole, so that the task with the longest overall path is scheduled first, thus effectively shortening the scheduling length

(2) The estimated remaining length table is defined based on the prescheduling results, which is more conducive to reflecting the real-time required from the current task to the exit task

(3) Select a processor for the task based on the predicted remaining length table. The selected processor may not be the earliest possible finish time for the current task, but it can make the finish time of the next phase task shorter and reduce the scheduling length

(4) The algorithm is evaluated from two aspects: randomly generated graphs and real-world application graphs. The experimental results show that our algorithm obtains a better scheduling length

The remainder of the paper is organized as follows. Section 2 describes the scheduling problem. Section 3 reviews the related work. The proposed algorithm is explained in Section 4. Experimental details and simulation results are presented in Section 5. And finally, we conclude in Section 6.

## 2. Task Scheduling Problem

An application can generally be represented by a directed acyclic graph (DAG) $G = (T, E)$, as shown in Figure 1, where

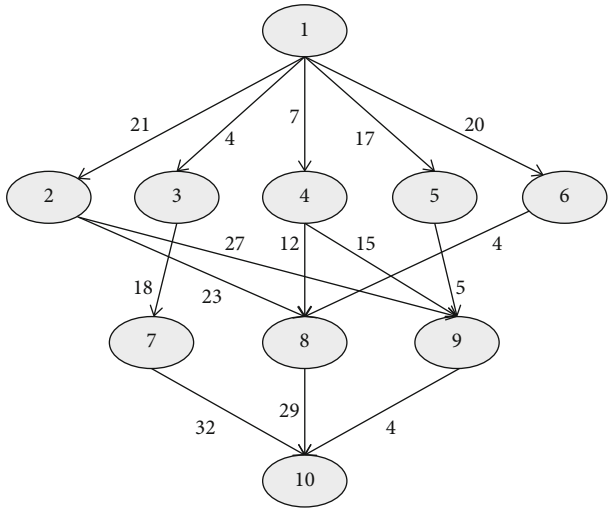


Figure 1: A sample DAG.

$T$ is a set of nodes and $E$ is a set of directed edges. Each node in the graph can be regarded as a task, and each edge represents the dependencies between tasks. For example, $e(i, j) \in E$ indicates that task $t_j$ is an immediate successor of task $t_i$, and task $t_i$ must finish its execution and transfer the resulting data to solve the data dependency before task $t_j$ starts. The weight of each edge $e(i, j) \in E$ represents the communication cost between task $t_i$ and $t_j$, denoted by $c_{i,j}$. The average communication cost of an edge $c_{i,j}$ is defined as

$$\bar{c}_{i,j} = \bar{L} + \frac{\text{data}_{ij}}{\bar{B}}, \tag{1}$$

where $\bar{L}$ is the average latency of all processors, $\bar{B}$ is the average bandwidth among processors, and $\text{data}_{ij}$ represents the amount of data to be transmitted from task $t_i$ to task $t_j$. If task $t_j$ and task $t_i$ are assigned to the same processor, $c_{i,j}$ becomes 0.

A matrix $W$ is generally applied as a supplement to the DAG. $W$ is a $t * p$ computation cost matrix, as shown in Table 1, where $t$ represents the task number, $p$ represents the processor number, and $w(t_i, p_j)$ represents the estimated execution time of task $t_i$ on processor $p_j$. The average execution time of task $t_i$ is defined as

$$\bar{w}_i = \frac{\left(\sum_{j=1}^{p} w\left(t_i, p_j\right)\right)}{p}. \tag{2}$$

Some basic concepts commonly used in task scheduling [35, 37, 38] are as follows:

*Definition 1.* $\text{pred}(t_i)$ and $\text{succ}(t_i)$ represent the set of immediate predecessors and the set of immediate successors of task $t_i$, respectively. If the $\text{pred}(t_i)$ or $\text{succ}(t_i)$ of the task $t_i$ is empty, it is called an entry task ($t_{\text{entry}}$) or an exit task ($t_{\text{exit}}$). For convenience, if there are more than 1 entry tasks (exit task) in a DAG, a dummy entry task (exit task) with 0 weights and 0 communication can be added to the graph.

*Definition 2 Earliest start time* (EST). The earliest start time of the task $t_i$ on the processor $p_j$ needs to satisfy all its parent tasks to be executed and the data transmitted to $t_i$, and the processor $p_j$ is in an idle state at this time. Therefore, the calculation formula of $\text{EST}(t_i, p_j)$ is as follows:

$$\text{EST}\left(t_i, p_j\right) = \max\left\{\text{avail}\left(p_j\right), \max_{t_k \in \text{pred}(t_i)}\left\{\text{AFT}(t_k) + \bar{c}_{i,k}\right\}\right\}, \tag{3}$$

where $\text{avail}(p_j)$ is the earliest available time of the processor, $\text{AFT}(t_k)$ represents the actual finish time of task $t_k$, and $\max_{t_k \in \text{pred}(t_i)}\{\text{AFT}(t_k) + \bar{c}_{i,k}\}$ denotes the arrival time of all input data for task $t_i$ on processor $p_j$.

*Definition 3 Earliest finish time* (EFT). The earliest finish time of task $t_i$ on processor $p_j$ is equal to the sum of the earliest start time and the execution time of task $t_i$ on processor $p_j$. The calculation formula of $\text{EFT}(t_i, p_j)$ is as follows:

$$\text{EFT}\left(t_i, p_j\right) = \text{EST}\left(t_i, p_j\right) + w_{i,j}. \tag{4}$$

*Definition 4 Total execution time or scheduling length* (makespan). The total execution time is the maximum value of the finish time of all exit tasks of the DAG. The calculation formula for makespan is as follows:

$$\text{makespan} = \max\left\{\text{AFT}(t_{\text{exit}})\right\}. \tag{5}$$

*Definition 5 Critical path* (CP). The critical path of DAG is the longest path from the entry node to the exit node. The lower bound of the scheduling length is the minimum critical path length (CPMIN), which is accumulated by the minimum execution cost of each task in the critical path.

*Definition 6 Average earliest start time* (AEST). The average earliest start time can be calculated recursively by traversing the DAG downward starting from the entry task. AEST calculation formula is as follows:

$$\text{AEST}(t_i) = \max_{t_k \in \text{pred}(t_i)}\left\{\text{AEST}(t_k) + \bar{w}_k + \bar{c}_{k,i}\right\}, \tag{6}$$

where $\text{AEST}(t_{\text{entry}}) = 0$.

*Definition 7 Average latest start time* (ALST). The average latest start time can be calculated recursively by traversing the DAG upward starting from the exit task. The ALST calculation formula is as follows:

Table 1: Computation time matrix of the tasks in each processor for a three-processor machine.

| Task | P1 | P2 | P3 |
|---|---|---|---|
| T1 | 36 | 17 | 35 |
| T2 | 11 | 17 | 22 |
| T3 | 21 | 19 | 35 |
| T4 | 31 | 14 | 21 |
| T5 | 14 | 14 | 17 |
| T6 | 19 | 15 | 10 |
| T7 | 13 | 20 | 20 |
| T8 | 29 | 14 | 30 |
| T9 | 5 | 11 | 11 |
| T10 | 24 | 20 | 27 |

$$\mathrm{ALST}(t_i) = \min_{t_k \epsilon \mathrm{succ}(t_i)} \left\{ \mathrm{ALST}(t_k) - \bar{c_{k,i}} \right\} - \bar{w_i}, \tag{7}$$

where $\mathrm{ALST}(t_{\mathrm{exit}}) = \mathrm{AEST}(t_{\mathrm{exit}})$.

## 3. Related Work

In cloud computing, efficient scheduling algorithms enable the system to process applications faster. How to perform application scheduling has a great impact on the system performance. Especially in a heterogeneous computing system, this greatly increases the complexity of the scheduling problem. Therefore, the task scheduling problem in heterogeneous computing systems has been concerned and researched all time. Such problems also belong to NP-hard problems. Therefore, many scheduling algorithms using heuristics have been researched to achieve the goal of reducing the scheduling length with lower time complexity. The most common one is the list scheduling algorithm. Compared with other clustering and duplication scheduling algorithms, its time complexity is relatively low and the scheduling results are better, so it is widely accepted and studied in task scheduling.

The list scheduling algorithm has two main phases: the prioritizing phase (the priority of each task is calculated) and the processor selection phase (the task is assigned an appropriate processor according to the scheduling strategy). The most typical representative is the Heterogeneous Earliest Finish Time (HEFT) algorithm proposed in the literature [35] and the Critical Path On a Processor (CPOP) [35] algorithm proposed together with the HEFT algorithm. However, neither of these two algorithms considers the impact of current task allocation on all successors of it. The lookahead algorithm in [40] uses the "lookahead" strategy to make up for the deficiency of HEFT in selecting the appropriate processor for the task and selects the appropriate processor by predicting the impact of the current task allocation on all successors, but the main disadvantage of this algorithm is its high time complexity. To overcome this issue, the PEFT algorithm has been proposed in [38]. It proposes an "Optimistic Cost Table" (OCT) to predict and reduce the time complexity, and the result also shows that the performance similar to the lookahead algorithm can be achieved. However, the PEFT algorithm ignores the influence of the current task execution cost on the priority ranking. For this reason, the literature [33] proposed the algorithm Predict Priority Task Scheduling (PPTS) based on the Predict Cost Matrix (PCM), and the experimental results show that the PPTS algorithm achieves better performance than PEFT. The drawback of the PPTS algorithm is that it does not remove the execution cost of the current task from the PCM value in the processor selection strategy, which will affect the choice of processor. In addition, [34, 36, 37, 41, 46] also proposed different list scheduling algorithms, which can also achieve good scheduling results. Next, the algorithms mentioned in the recently proposed IPPTS [34] and literature [46] will be introduced in detail, which will also be used as a comparison algorithm for the algorithms proposed in this paper.

The IPPTS [34] algorithm is an improved algorithm based on the PPTS algorithm. The algorithm calculates the task priority by multiplying the average PCM value of the task by the number of its immediate successors. The purpose is to give higher weight to tasks with more immediate successors so that more tasks enter the ready list. In the processor selection phase, the processor that minimizes the sum of task finish time and "the Looking Head Exit Time" (LHET) is selected to realize "upward" and "downward" forecasting. Literature [34] also proved through experiments that the IPPTS algorithm can achieve better performance than the same type of algorithms such as HEFT, CPOP, PEFT, IPEFT [37], and PPTS. It also mentions that the algorithm does not perform well for the application graph algorithm with more critical tasks.

The authors in literature [46] proposed a new list scheduling algorithm (referred to as RE-18 in this paper). It is divided into three phases: the level sorting phase, the task prioritization phase, and the processor selection phase. The purpose of the level sorting phase is to determine the dependencies of tasks to determine the level of the task. In the task prioritization phase, three attributes of the current task's cumulative execution cost (CEC), data transfer cost (DTC), and rank of predecessor task (RPT) are considered. The purpose of it is to assign higher priority to tasks with more immediate successor tasks as much as possible. In the processor selection phase, a noncrossover technique [53] is used to select the processor, and the task is executed on the processor with small EFT or the processor with the least execution cost. Literature [46] proved through experiments that the proposed RE-18 algorithm has better performance than HEFT, PEFT, and other algorithms. Although the RE-18 algorithm uses a noncrossover technique to reduce the execution cost of tasks, it may also increase the communication cost and cause performance degradation.

## 4. The Proposed Algorithm BVSL

In this part, the proposed BVSL algorithm will be introduced in detail through five subsections: virtual scheduling length table, prescheduling phase, task prioritization phase,

processor selection phase, and detailed description of the BVSL algorithm.

*4.1. Virtual Scheduling Length Table.* This section will propose a "Virtual Schedule Length Table" (VSLT), which determines the priority ordering of tasks.

In the list scheduling algorithm, the priority ranking calculations of many algorithms are improved or expanded based on the $\text{rank}_u$ ($\text{rank}_u(t_i) = \bar{w}_i + \max_{t_k \epsilon \text{succ}(t_i)} \{\bar{c_{i,j}} + \text{rank}_u(t_k)\}$) calculation method of the HEFT algorithm. This type of calculation method only considers the execution cost of the current task, its successor tasks, and the communication cost between tasks. Therefore, it cannot well reflect the real time required from the current task to the exit task. For this reason, we use the existing list scheduling algorithm to perform prescheduling to obtain a real scheduling result, and then, the predicted remaining length table (PRLT) is constructed based on the real scheduling result. PRLT is a matrix in which each $\text{PRLT}(t_i, p_j)$ represents the predicted remaining length of task $t_i$ on processor $p_j$, that is, the length from task $t_i$ to the exit task when task $t_i$ is on processor $p_j$. $\text{PRLT}(t_i, p_j)$ is calculated as

$$\text{PRLT}\left(t_i, p_j\right) = \text{premakespan} - \min_{t_k \epsilon \text{succ}(t_i)} \left\{\text{preAST}_{(t_k)} - \bar{c_{i,k}}\right\}, \tag{8}$$

where $\text{preAST}_{(t_k)}$ is the actual start time of the task $t_k$ in the prescheduling and $\min_{t_k \epsilon \text{succ}(t_i)} \{\text{preAST}_{(t_k)} - \bar{c_{i,k}}\}$ represents the latest finish time of the task $t_i$ on the processor $p_j$. If the task $t_k$ is assigned the same processor as $p_j$ in the prescheduling, $\bar{c_{i,k}} = 0$. For exit tasks, $\text{PRLT}(t_{\text{exit}}, p_j) = 0$.

In addition, due to the difference in the start execution time of the ready task, the priority of the task may be affected (an example will be given in Section 4.5). Hence, we propose VSLT to overcome this drawback. VSLT is defined as

$$\text{VSLT}\left(t_i, p_j\right) = \text{EST}\left(t_i, p_j\right) + w\left(t_i, p_j\right) + \text{PRLT}\left(t_i, p_j\right), \tag{9}$$

where the first part of the formula represents the earliest start execution time of task $t_i$ on processor $p_j$, the middle part represents the estimated execution time of task $t_i$ on processor $p_j$, and the last part $\text{PRLT}(t_i, p_j)$ represents the predicted remaining length of task $t_i$ on processor $p_j$. The sum of these three parts can be regarded as the total estimated scheduling length when only the currently ready task is considered, which helps us to analyze the impact of the current task on the scheduling length as a whole.

Table 2 shows the comparison of better and worse results when VSLT and PRLT are used to consider priority, respectively (equal results are removed). It can be seen that for different types of application graphs, the results of using VSLT to consider priority sorting are nearly 20% better than those of using PRLT to consider priority sorting and reach the highest 35.6% in the Montage application graph. This shows that using VSLT to consider prioritization can improve the performance of the algorithm to a certain extent. Therefore, VSLT will be used as the basis for prioritization in the proposed algorithm (the application graph of the experiment here is the same as the one used in part 5).

Table 2: Comparison of better and worse results when VSLT and PRLT are used to consider priority, respectively (equal results are removed).

| Our algorithm | Metrics | PRSL | | | |
|---|---|---|---|---|---|
| | | Montage | Epigenomics | SIPHT | Random DAG |
| VSLT | Better (%) | 67.8 | 63.8 | 59 | 60 |
| | Worse (%) | 32.2 | 36.2 | 41 | 40 |

*4.2. Prescheduling Phase.* To make the sorting results of tasks closer to the real scheduling, we add a prescheduling phase before the task selection phase. In this phase, we choose the PPTS algorithm as the prescheduling algorithm. The $\text{preMap} < t_i, p_j >$, $\text{preAST}(t_i)$, and premakespan generated by the prescheduling will be used as the input for the next phase, where $\text{preMap} < t_i, p_j >$ represents the result of mapping between tasks and processors in prescheduling, $\text{preAST}(t_i)$ represents the actual start time of tasks in prescheduling, and premakespan represents the scheduling length when prescheduling is completed.

*4.3. Task Prioritization Phase.* To prioritize tasks, the average VSLT will be calculated for each task. The calculation formula is as follows:

$$\text{rank}_{\text{VSLT}}(t_i) = \frac{1}{p} \sum_{j=1}^{p} \left(\text{VSLT}\left(t_i, p_j\right)\right). \tag{10}$$

In order to allow tasks with a larger average VSLT value to be executed firstly, the task priorities are sorted in descending order according to the $\text{rank}_{\text{VSLT}}(t_i)$ value.

*4.4. Processor Selection Phase.* To select the processors for a task, the earliest finish time of the task on each processor is first calculated. The insertion-based policy is applied to compute EFT, that is, the possibility of inserting tasks in the earliest idle time slot between 2 scheduled tasks on the identical processor should be considered. The idle time slot should be at least capable of handling the computation cost of the task to be scheduled, and scheduling on this idle time slot should preserve precedence constraints.

Then, the $\text{EFT}_{\text{PRLT}}$ of the task on each processor is calculated. $\text{EFT}_{\text{PRLT}}$ is defined as

$$\text{EFT}_{\text{PRLT}}\left(t_i, p_j\right) = \text{EFT}\left(t_i, p_j\right) + \text{PRLT}\left(t_i, p_j\right). \tag{11}$$

Finally, the processor with the minimum $EFT_{PRLT}$ value is selected for the current task. Although the finish time of the chosen processor is not always the earliest, this processor selection policy not only considers the EFT value of the current task but also considers the impact of the chosen processor on the path length from the current task to the exit nodes. Therefore, the scheduling length can be effectively reduced to a certain extent.

*4.5. Detailed Description of the BVSL Algorithm.* The pseudo-code of the algorithm is shown in Algorithm 1. First, the algorithm uses the PPTS algorithm for prescheduling and then calculates the PRLT based on the prescheduling result (lines 1 and 2). Then, an empty ready list is created, and the entry task is placed on top of the list (line 3). In the loop of while, it firstly calculates the values of EST, VSLT, and $rank_{VSLT}$ for all ready-list tasks and then selects the task with the highest $rank_{VSLT}$ value as the currently scheduled task (lines 5 and 6). After selecting the task for scheduling, the $EFT_{PRLT}$ values for the task on all processors are calculated and the processor $p_j$ with the minimum $EFT_{PRLT}$ is selected to execute task $t_i$ (lines 7-11). Finally, return the better result of the scheduling result and the prescheduling result as the final scheduling result (lines 14-18).

The time complexities of each step in the algorithm are as follows:

(1) The time complexity of the prescheduling with the PPTS algorithm is $O(t^2 * p)$, where $p$ is the number of processors and $t$ is the number of nodes in the DAG

(2) The time complexity of calculating PRLT is $O(p * (t + e))$, where $t$ is the number of nodes in the DAG

(3) The time complexity of calculating VSLT and processor selection is $O(t^2 * p)$

(4) The time complexity of calculating $rank_{VSLT}$ is $O(t * p)$

Therefore, the total time complexity of the algorithm is $O(t^2 * p)$.

Tables 3 and 4, respectively, show the results of DAG prescheduling and the PRLT values of tasks on different processors in Figure 1. Table 5 shows the process of selecting processors in each iteration of the algorithm. Figure 2 shows the schedule of the example task graph in Figure 1 using the BVSL algorithm, the scheduling algorithm based on average PRLT to calculate task priority, IPPTS algorithm, and RE-18 algorithm. The makespan of the BVSL algorithm is 105, which is shorter than that of other algorithms.

For the DAG in Figure 1, the sorted list obtained by using the average PRLT sorting is (T1, T3, T2, T4, T6, T5, T7, T8, T9, T10), and the sorted list obtained by using the average VSLT sorting is (T1, T3, T2, T4, T6, T8, T5, T7, T9, T10). Compared with the former, the latter schedules the task T8 before tasks T5 and T7. The average PRLT of T8 is smaller than that of tasks T5 and T7, while its average VSLT value is larger than that of T5 and T7. As can be seen from Figure 2 that the only difference between Figures 2(a) and 2(b) is that task T8 is allocated before task T7 in processor P2, but the final scheduling length is shorter than that in Figure 2(b). Therefore, the start time of the task will also affect the priority of the task to a certain extent, resulting in a difference in the final result.

## 5. Experimental Results and Discussion

This section will introduce the comparison between the BVSL algorithm and the latest IPPTS and RE-18 algorithms. First, some comparison metrics will be introduced.

*5.1. Comparison Metrics.* The comparison of the algorithms in this paper is based on the following four metrics:

(1) Total execution time or scheduling length (makespan) [46]

Makespan is the time required for an application from the execution of the first task to the end of the last task. One of the goals of the list scheduling algorithm is to minimize the makespan. The calculation is shown in Definition 4.

(2) Schedule length ratio (SLR) [35, 37]

The scheduling length ratio (SLR) is the normalization of the schedule length, and the definition formula is as follows:

$$\mathrm{SLR} = \frac{\mathrm{makespan(solution)}}{\sum_{t_i \in CP_{\mathrm{MIN}}} \min_{p_j \in P} \left( w\left(t_i, p_j\right)\right)}. \tag{12}$$

The denominator in the above formula is the minimum computation cost (CPMIN) of the task on the critical path. A lower SLR indicates a superior algorithm.

(3) Number of occurrences of better quality of schedules (NOBQSs) [35, 37]

NOBQS represents the percentage of occurrences of better, equal, and worse scheduling lengths between the two algorithms.

(4) Speedup [35]

Speedup is the ratio of the sequential execution time to the parallel execution time (i.e., makespan). Therefore, the larger the speedup, the better the scheduling algorithm. The sequential execution time is calculated by assigning all tasks to a single processor that minimizes the total computation cost of the task graph. The speedup calculation formula is as follows:

Input: Workflow $G(T,E)$; a set $Q$ of $P$ heterogeneous processors
Output: Schedule=$\{t_i, p_j, \mathrm{EST}(t_i, p_j), \mathrm{EFT}(t_i, p_j)\}$
1. Execute the pre-scheduling PPTS algorithm
2. Calculate the PRLT using pre-scheduling results for all tasks
3. Create an Empty List ready-list and put $t_{\mathrm{entry}}$ as the initial task
4. While ready-list is not empty do
5. Compute the EST, VSLT and $\mathrm{rank}_{\mathrm{VSLT}}$ for all ready-list tasks
6. $t_i \longleftarrow$ the task with the highest $\mathrm{rank}_{\mathrm{VSLT}}$ from ready-list
7. For each processor $p_j$ in the processor set $P$ do
8. Compute the $\mathrm{EFT}(t_i, p_j)$ value using the insertion-based scheduling policy
9. Compute $\mathrm{EFT}_{\mathrm{PRLT}}$
10. End for
11. Assign task $t_i$ to the processor $p_j$ that minimize
12. $\mathrm{EFT}_{\mathrm{PRLT}}$ of task $t_i$
13. Update ready-list
14. End while
15. If scheduling results are worse than pre-scheduling results
16. Return pre-scheduling results
17. else
18. Return scheduling results
19. End if

Algorithm 1: Pseudo-code of BVSL algorithm.

Table 3: Prescheduling results.

| Task | preMap | preAST | premakespan |
|---|---|---|---|
| T1 | P2 | 0 | 119 |
| T2 | P2 | 36 | |
| T3 | P2 | 17 | |
| T4 | P3 | 24 | |
| T5 | P1 | 34 | |
| T6 | P3 | 45 | |
| T7 | P1 | 54 | |
| T8 | P2 | 59 | |
| T9 | P2 | 73 | |
| T10 | P2 | 99 | |

Table 4: PRLT of DAG in Figure 1.

| Task | P1 | P2 | P3 | AVG |
|---|---|---|---|---|
| T1 | 106 | 102 | 106 | 104.7 |
| T2 | 83 | 60 | 83 | 75.3 |
| T3 | 65 | 83 | 83 | 77 |
| T4 | 72 | 60 | 72 | 68 |
| T5 | 51 | 46 | 51 | 49.3 |
| T6 | 64 | 60 | 64 | 62.7 |
| T7 | 52 | 20 | 52 | 41.3 |
| T8 | 49 | 20 | 49 | 39.3 |
| T9 | 24 | 20 | 24 | 22.7 |
| T10 | 0 | 0 | 0 | 0 |

$$\mathrm{Speedup} = \frac{\min_{p_j \in P}\left\{\sum_{t_i \in T} w\left(t_i, p_j\right)\right\}}{\mathrm{makespan}}. \tag{13}$$

### 5.2. Randomly Generated Application Graphs

*5.2.1. Random Graph Generator.* Our random graph will be generated using the task graph generator https://github.com/wzwtime/RandomGraphGenerator_new. The following parameters need to be used [34, 35]:

(1) Number of tasks in the graph ($t$)

(2) *The Shape Parameter of the Graph ($\alpha$).* We assume that the height (depth) of the DAG is randomly generated from a uniform distribution with an average value equal to $\sqrt{t}/\alpha$ (the height is equal to the smallest integer value that is not less than the randomly generated actual value). The width of each level is randomly selected from a uniform distribution with an average value equal to $\alpha * \sqrt{t}$. If the selected $\alpha$ value is greater than 1, a dense graph (shorter graph with a high degree of parallelism) is generated. If the selected $\alpha$ value is less than 1, a longer graph with a lower degree of parallelism is generated.

(3) Out degree of a node, (out_degree).

(4) *Communication to Computation Ratio (CCR).* It is the ratio of the average communication cost to the average computational cost among all tasks.

Table 5: The scheduling result generated by the BVSL algorithm in each iteration.

| Step | Ready-list and $rank_{VSLT}$ | Task select | EFT | | | $EFT_{PRLT}$ | | | CPU select |
|---|---|---|---|---|---|---|---|---|---|
| | | | P1 | P2 | P3 | P1 | P2 | P3 | |
| 1 | T1(134) | T1 | 36 | 17 | 35 | 142 | 119 | 141 | P2 |
| 2 | T2 (109), T3 (119), T4 (107), T5 (81.3), T6 (94.3) | T3 | 42 | 36 | 56 | 107 | 119 | 139 | P1 |
| 3 | T2 (117.3), T4 (115.3), T5 (89.7), T6 (102.7), T7 (101) | T2 | 53 | 34 | 60 | 136 | 94 | 143 | P2 |
| 4 | T4 (121), T5 (95.3), T6 (108.3), T7 (101) | T4 | 73 | 48 | 45 | 145 | 108 | 117 | P2 |
| 5 | T5 (100), T6 (113), T7 (103) | T6 | 61 | 63 | 47 | 125 | 123 | 111 | P3 |
| 6 | T5 (110), T7 (104.7), T8 (111.7), | T8 | 89 | 65 | 90 | 138 | 85 | 139 | P2 |
| 7 | T5 (115.7), T7 (110.3) | T5 | 56 | 79 | 64 | 107 | 125 | 115 | P1 |
| 8 | T7 (115), T9 (90.7) | T7 | 69 | 85 | 80 | 121 | 105 | 132 | P2 |
| 9 | T9 (97.3) | T9 | 68 | 96 | 74 | 92 | 116 | 98 | P1 |
| 10 | T10 (108.7) | T10 | 141 | 105 | 144 | 141 | 105 | 144 | P2 |

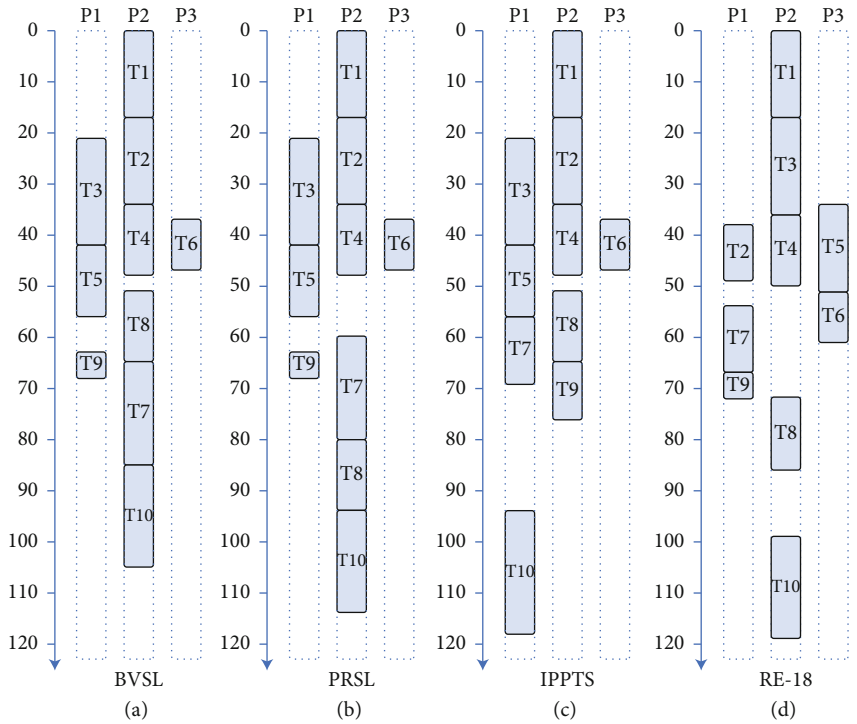


Figure 2: Schedules of the sample DAG in Figure 1 with (a) BVSL (makespan = 105), (b) prioritization using average PRLT (makespan = 114), (c) IPPT (makespan = 118), and (d) RE-18 (makespan = 119).

(5) *Range Percentage of Computation Cost on Processors ($\beta$).* It is a heterogeneous factor of processor speed. A high $\beta$ value will result in a significant difference in the computation costs of tasks between processors, and a low $\beta$ value indicates that the computation costs of a given task are almost equal among processors. The average computation costs of each task in the graph are randomly selected from a uniform distribution with the range $[0, 2 * \bar{w}_{DAG}]$, where $\bar{w}_{DAG}$ is the average computational cost of a given graph that is obtained randomly. The computing cost of the task $t_i$ on each processor $p_j$ is randomly set within the following range:

$$\bar{w}_i \times \left(1 - \frac{\beta}{2}\right) \leq w\left(t_i, p_j\right) \leq \bar{w}_i \times \left(1 + \frac{\beta}{2}\right) \quad (14)$$

In the experiment, we consider the following parameters to generate a random graph.

(i) $t = \{20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400\}$

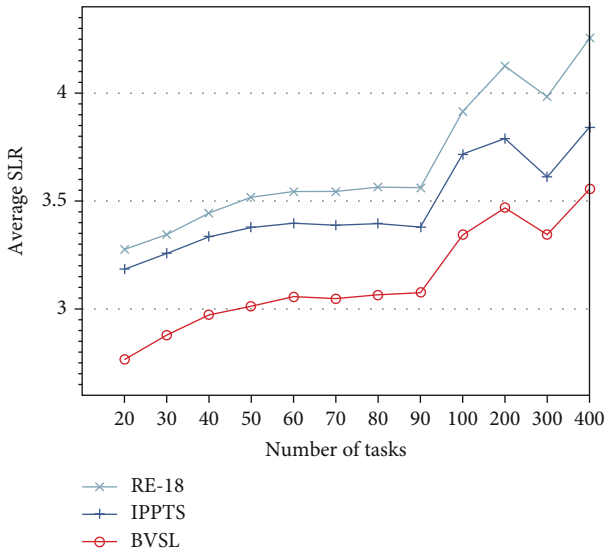


Figure 3: Average SLR under different number of tasks on random application graph.

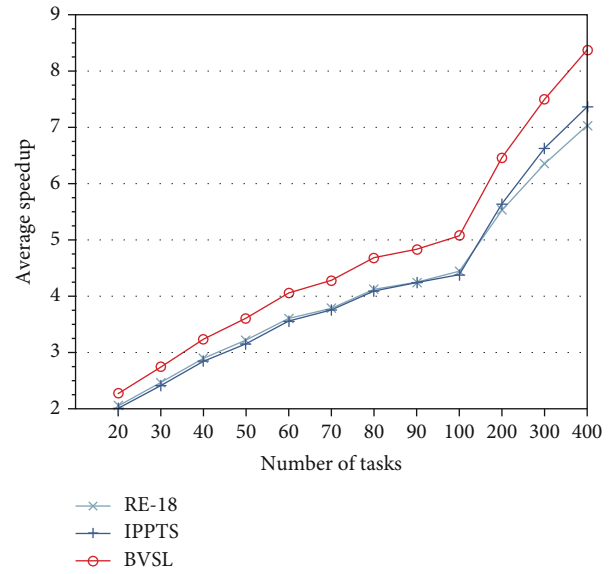


Figure 4: Average speedup under different number of tasks on random application graph.

(ii) CCR = {0.1, 0.5, 1, 2, 5, 10}

(iii) $\beta$ = {0.1, 0.2, 0.5, 0.75, 1, 2}

(iv) Processors = {4, 8, 16, 32}

(v) $\alpha$ is randomly selected from {1, 2}

(vi) out_degree is randomly selected from {1, 2, 3, 4, 5}

These combinations produce a total of 1728 different DAG types, and 30 random DAGs will be generated for each type of DAG, so there are 51840 DAGs in our experiment.

*5.2.2. Performance Results.* Figure 3 shows the average SLR results of all algorithms under the different number of tasks. When the number of tasks is 20, the average SLR of BVSL is 15.6% lower than RE-18 and 13% lower than IPPTS, and when the number of tasks reaches 400, the average SLR of BVSL is 7.5% lower than that of IPPTS. This implies that the advantages of BVSL compared to IPPTS decrease for an increasing number of tasks.

Figure 4 shows the average speedup for all algorithms as a function of the DAG size. For DAGs with 20 tasks, the BVSL algorithm outperforms the RE-18 algorithm by 10.7 percent and the IPPTS algorithm by 13% percent. For DAGs with 400 tasks, BVSL outperforms RE-18 by 19 percent and IPPTS by 13 percent.

Figure 5 presents the average SLR of the algorithms for heterogeneity values of [0.1, 0.2, 0.5, 0.75, 1, 2]. When $\beta$ = 0.1, the BVSL algorithm is 9.4% better than RE-18 and 4% better than IPPTS. When $\beta$ = 2, the BVSL algorithm is 19.4% better than RE-18 and 15.6% better than IPPTS.

The average SLR produced by the algorithms as a function of CCR is shown in Figure 6. When CCR = 10, the BVSL algorithm is 14.5% and 8.8% better than RE-18 and IPPTS, respectively.

The average makespan as function on the number of processors is shown in Figure 7. When the number of

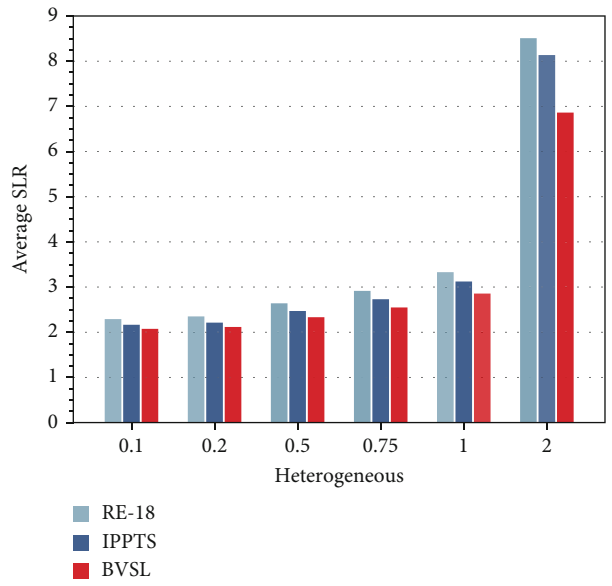


Figure 5: Average SLR of random application graph under different heterogeneous.

processors is 4, the average makespan of the BVSL algorithm is 13.8% lower than RE-18 and 6.3% lower than IPPTS. When the number increases to 32, the average makespan of the BVSL algorithm is 12.8% lower than that of RE-18 and 6.0% lower than that of IPPTS.

Figure 8 shows the average makespan of the algorithms with respect to the heterogeneity values. For a heterogeneity value of 0.1, the BVSL algorithm obtains the best results, with an average improvement of 10 percent and 2.9 percent over RE-18 and IPPTS, respectively. For a heterogeneity value of 2, the improvement over RE-18 and IPPTS increases to 19% and 15.5%, respectively. It can be seen that the BVSL

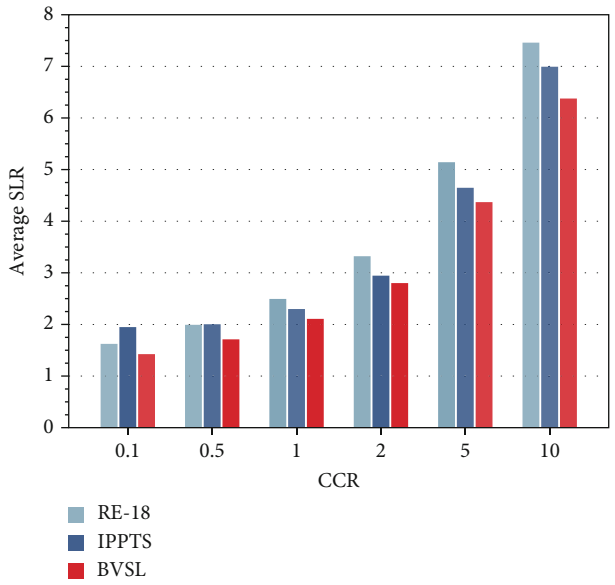


Figure 6: Average SLR of random application graph under different CCR.

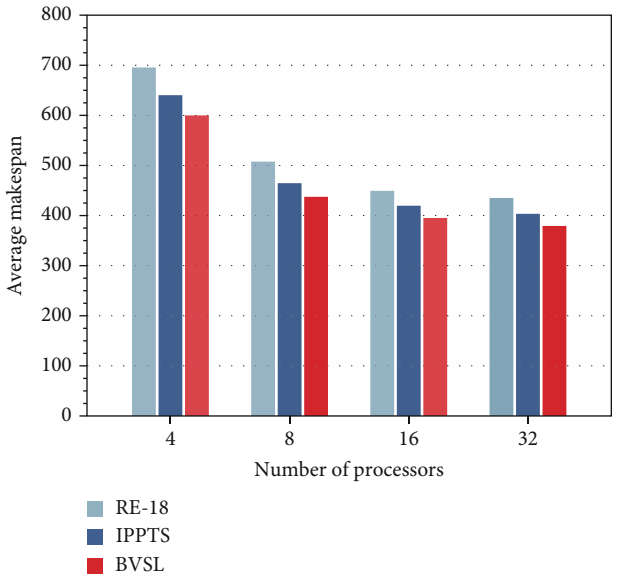


Figure 7: Average makespan of random application graph under different number of processors.

algorithm can achieve better results in a highly heterogeneous situation.

Table 6 lists the percentages of better, equal, and worse scheduling lengths generated by each algorithm compared with the remaining algorithms. Compared with the IPPTS and RE-18 algorithms, the BVSL algorithm achieves better scheduling in 78.5% and 91.3% of runs, equivalent schedules in 5.8% and 1.4% of runs, and worse schedules in 15.7% and 7.3% of runs.

*5.3. Real-World Application Graphs.* In addition to randomly generated application graphs, we also considered three real-

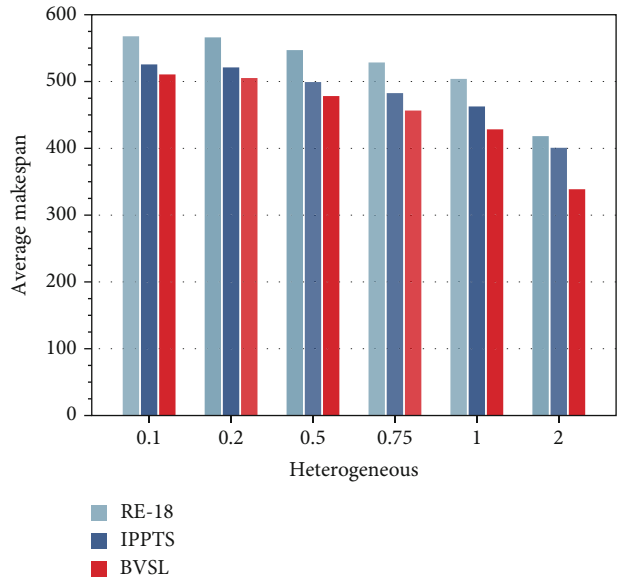


Figure 8: Average makespan of random application graph under different heterogeneous.

Table 6: Pairwise schedule length comparison of the scheduling algorithms.

| Algorithm | Metrics | Algorithm BVSL (%) | Algorithm IPPTS (%) | Algorithm RE-18 (%) |
|---|---|---|---|---|
| BVSL | Better | * | 78.5 | 91.3 |
| | Worse | * | 15.7 | 7.3 |
| | Equal | * | 5.8 | 1.4 |
| IPPTS | Better | 15.7 | * | 71.6 |
| | Worse | 78.5 | * | 26.8 |
| | Equal | 5.8 | * | 1.6 |
| RE-18 | Better | 7.3 | 26.8 | * |
| | Worse | 91.3 | 71.6 | * |
| | Equal | 1.4 | 1.6 | * |

world application graphs: Montage [54–56], Epigenomics [55], and SIPHT [55]. Because of the known structure of these three workflows, we simply used different values for CCR, heterogeneity, and number of processors. The range of values that we used in our simulation are set as follows:

(i) CCR = {0.1, 0.5, 1, 2, 5, 10}

(ii) $\beta$ = {0.1, 0.2, 0.5, 0.75, 1, 2}

(iii) Processors = {4, 8, 16, 32}

*5.3.1. Montage.* We chose Montage graphs with 25, 50, and 100 tasks. The average SLRs for different levels of heterogeneity and different CCRs are illustrated in Figures 9 and 10. The BSLV algorithm outperformed RE-18 by 12.4 and 21 percent and IPPTS by 8.8 and 19.5 percent for a $\beta$ value of 0.1 and 2, respectively. In terms of CCR, the BVSL and

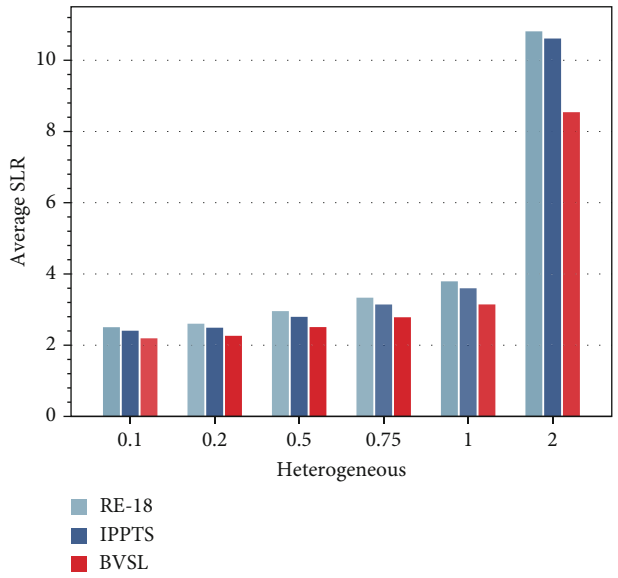


Figure 9: Montage application graph average SLR under different heterogeneous.

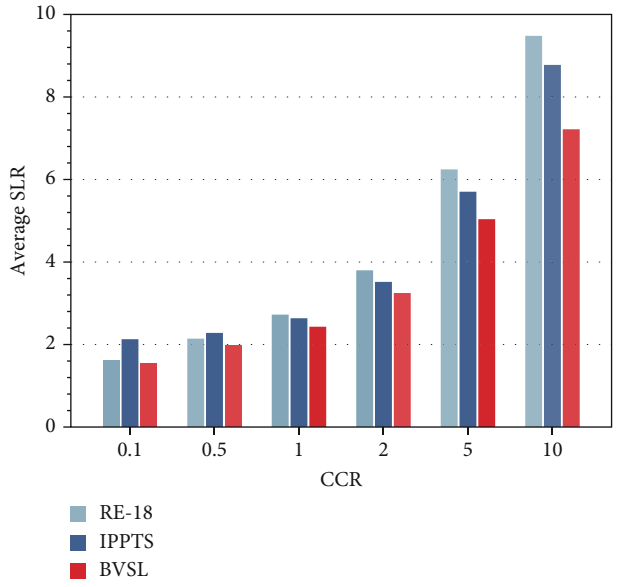


Figure 10: Montage application graph average SLR under different CCR.

RE-18 algorithms present similar performance for a low CCR value of 0.1. When CCR = 10, BSVL surpassed RE-18 and IPPTS by 23.9 and 17.8 percent, respectively.

Figure 11 shows the average speedup under different heterogeneous conditions. When $\beta = 2$, the BSLV algorithm is 11% better than RE-18 and 54% better than IPPTS. In terms of CCR (Figure 12), the average SLR improvement of BSLV over RE-18 and IPPTS is 26.5% and 19% for CCR = 10, respectively.

*5.3.2. Epigenomics.* Secondly, for the Epigenomics application graph, we chose the application graphs of 24, 46, and 100 task sizes. The results are shown in Figures 13–16. The

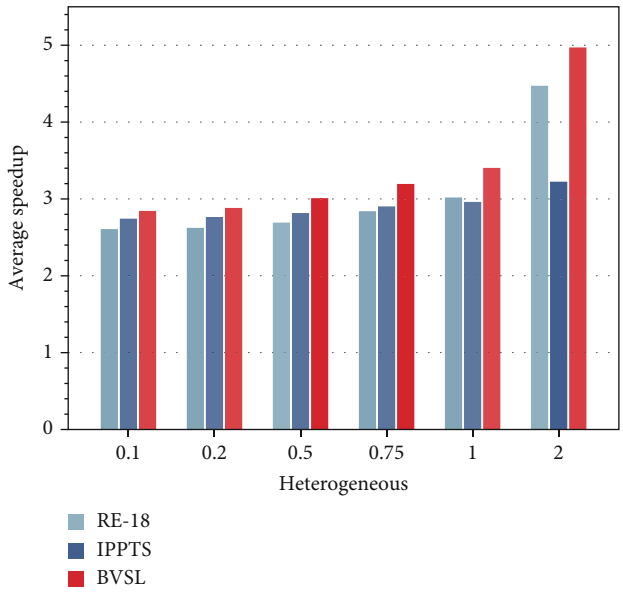


Figure 11: Montage application graph average speedup under different heterogeneous.

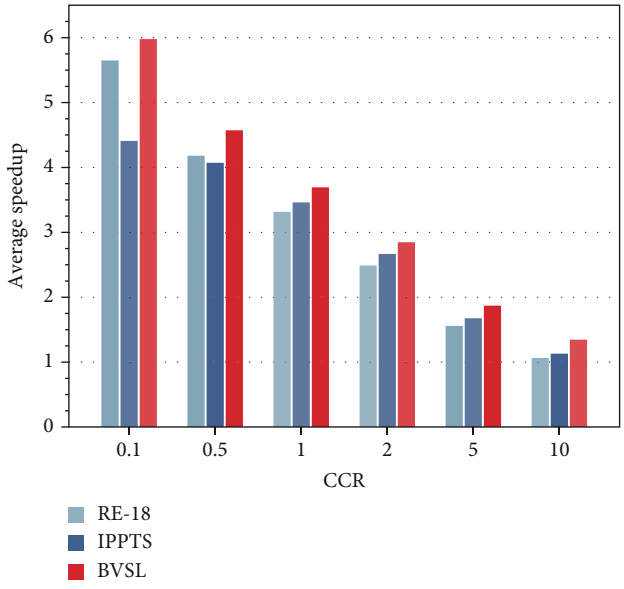


Figure 12: Montage application graph average speedup under different CCR.

BVSL algorithm maintains the best performance overall. The average SLRs obtained for the BVSL, RE-18, and IPPTS algorithms as a function of CCR and heterogeneity are shown in Figures 13 and 14, respectively. For different heterogeneous, starting from $\beta = 0.1$ to $\beta = 2.0$, the improvement of BVSL compared with RE-18 increased from 6.5% to 15.8%, and the improvement of BVSL compared with IPPTS increased from 4.4% to 13.7%. For different CCRs, the average SLR of BSLV algorithm is also lower than RE-18 and IPPTS. When CCR = 10, BSLV is improved by 12.2% compared with RE-18 and 10.2% compared with IPPTS.

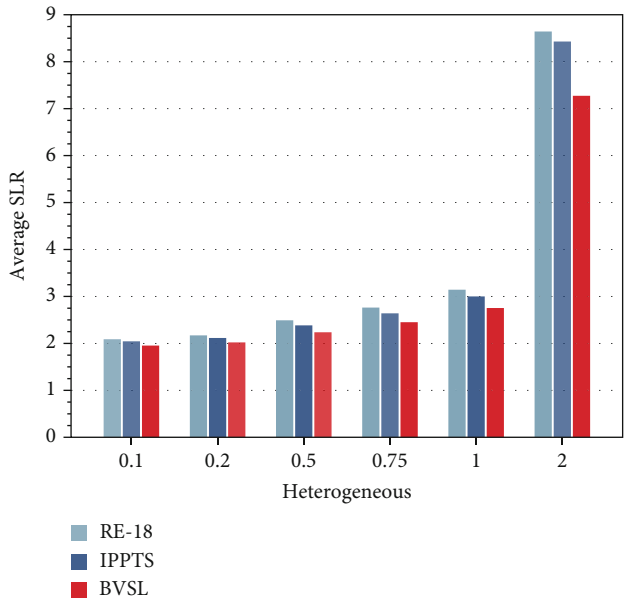


Figure 13: Epigenomics application graph average SLR under different heterogeneous.

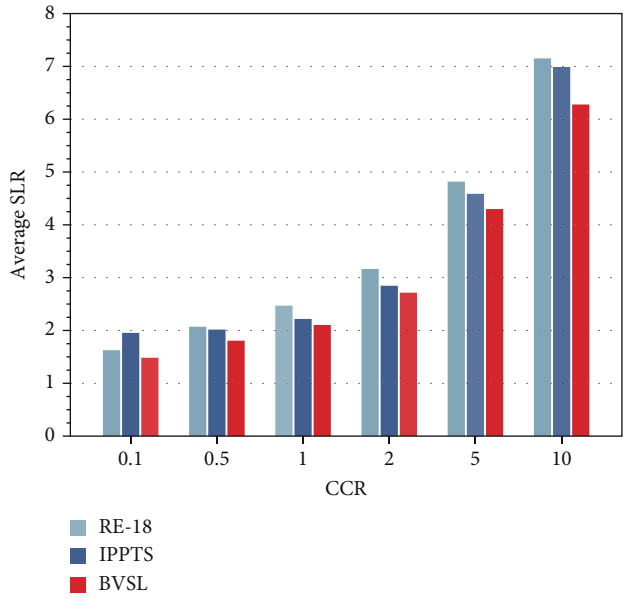


Figure 14: Epigenomics application graph average SLR under different CCR.

The average makespan for different heterogeneities and different CCRs are illustrated in Figures 15 and 16, respectively. When $\beta = 2$, the average makespan of BVSL reduces by 15.7% compared with RE-18 and 14% compared with IPPTS. For CCR = 10, the average makespan of BVSL performs better than RE-18 and IPPTS by 10.7% and 10%, respectively.

*5.3.3. SIPHT.* In SIPHT, application graphs of 30, 60, and 100 task sizes were selected. The average SLRs for different

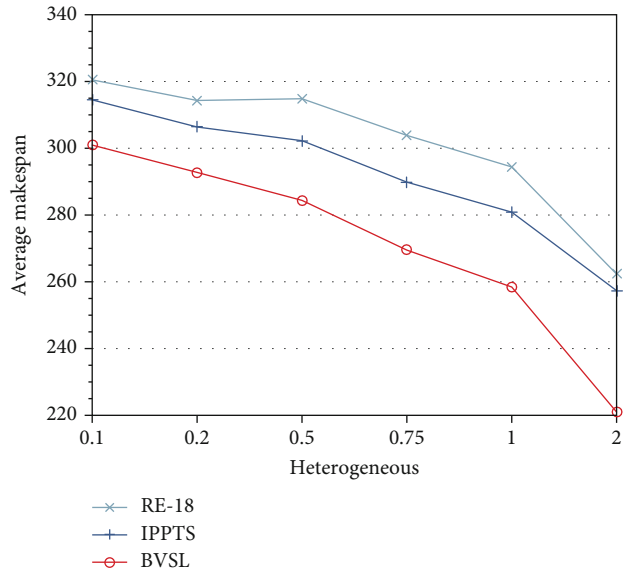


Figure 15: Epigenomics application graph average makespan under different heterogeneous.

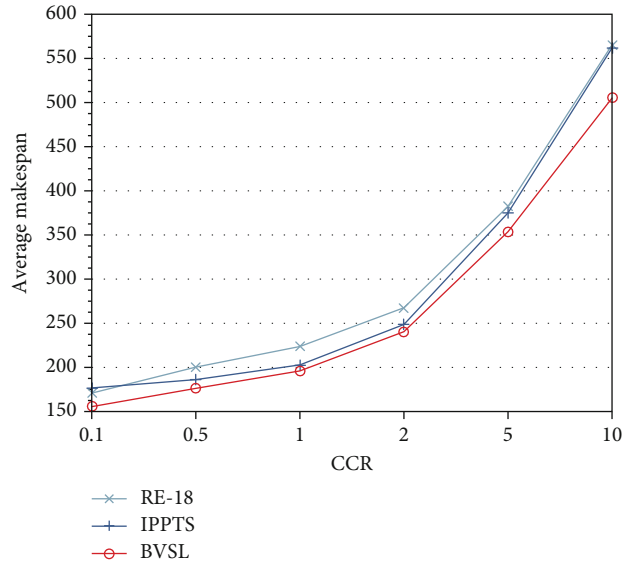


Figure 16: Epigenomics application graph average makespan under different CCR.

heterogeneities and different CCRs are shown in Figures 17 and 18, respectively. For average SLRs, the performance of the BVSL algorithm is always better than other algorithms. In Figure 17, when $\beta = 2$, the BVSL algorithm is 24.9% better than RE-18 and 27.4% better than IPPTS. In Figure 18, when CCR = 10, the BVSL algorithm is 32.9% better than RE-18 and 29.7% better than IPPTS.

Figure 19 shows the average speedup under different heterogeneous conditions. When $\beta = 2$, the BSLV algorithm is 14.4% better than RE-18 and 68.8% better than IPPTS. Figure 20 shows the average speedup under different CCRs. When CCR = 10, the BSLV algorithm is 47.3% better than

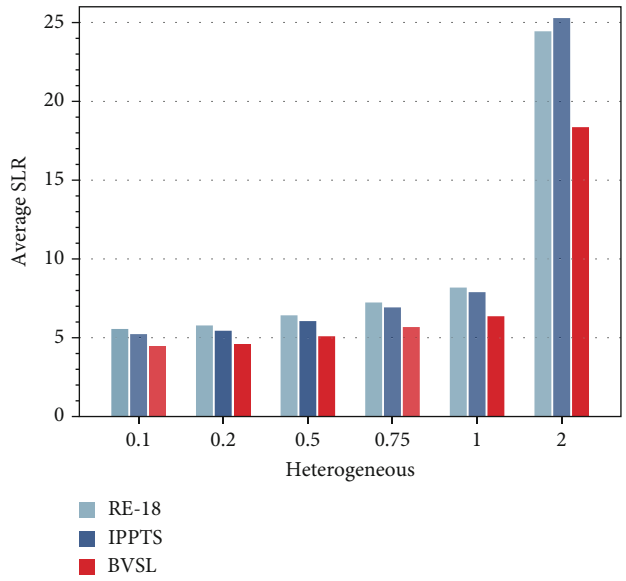


Figure 17: SIPHT application graph average SLR under different heterogeneous.

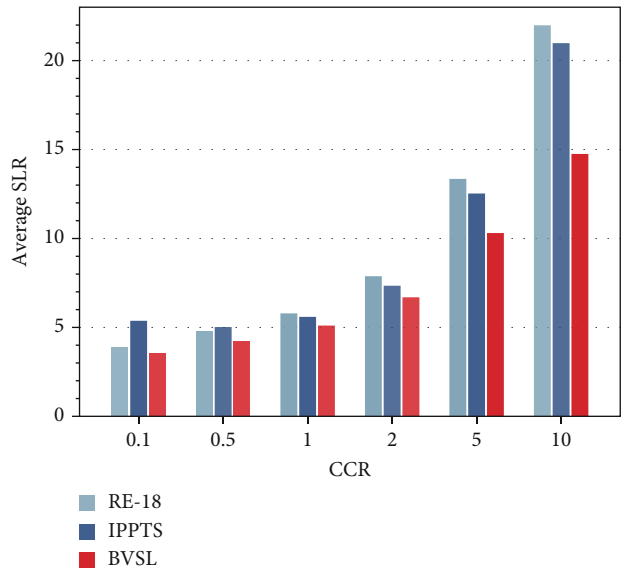


Figure 18: SIPHT application graph average SLR under different CCR.

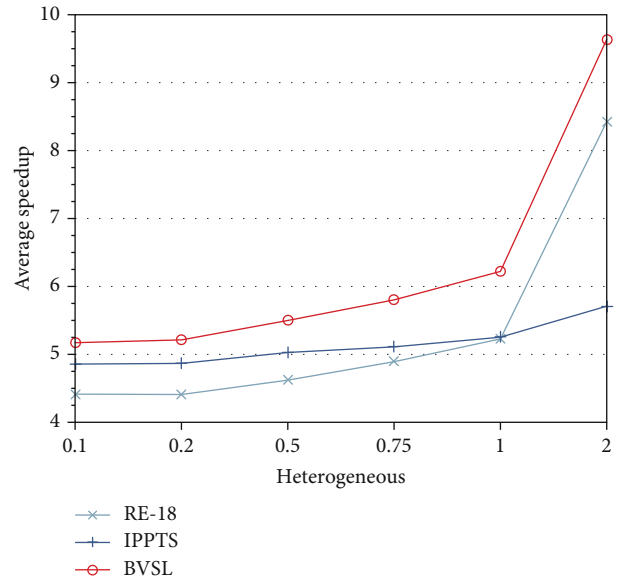


Figure 19: SIPHT application graph average speed under different heterogeneous.

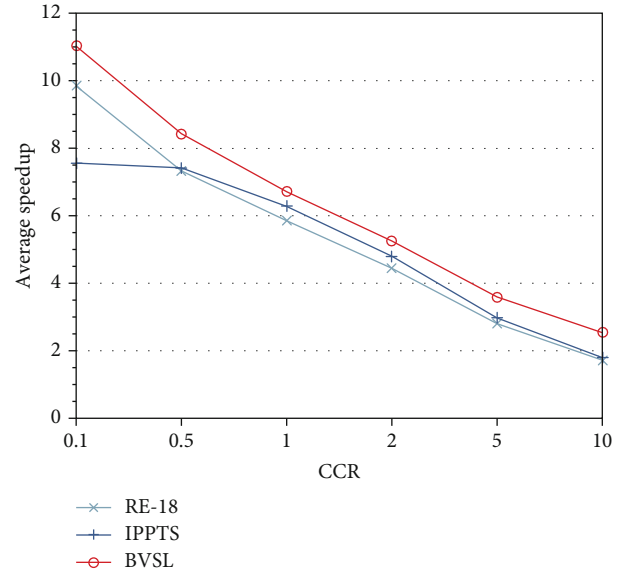


Figure 20: SIPHT application graph average speed under different CCR.

RE-18 and 41% better than IPPTS. In addition, IPPTS has poor performance with high heterogeneity and low CCR.

From the above experiments, we can see that the algorithm proposed in this paper has better performance, especially in the case of high heterogeneity. The main reason is that the higher the heterogeneity, the difference in the earliest start time of ready tasks will also increase. IPPTS algorithm uses bottom-up to calculate the priority, which will ignore the impact of this difference. RE-18 uses the top-down method to calculate the priority, and the priority of ready tasks will depend on their parent tasks. At this time, the higher the heterogeneity, the lower the fitness of the original priority order for the continuous scheduling of tasks. The BVSL algorithm realizes the overall consideration of task priority through prescheduling, which makes up for the above defects and improves the performance to a certain extent.

## 6. Conclusions

In general, resource management and task scheduling in edge computing and cloud computing are important factors to improve the performance of the system, and the same is true in cloud edge collaboration. This paper proposes a

new list scheduling algorithm BVSL based on a virtual scheduling length table for heterogeneous computing systems in cloud. The algorithm first uses the PPTS algorithm for prescheduling and calculates the predicted remaining length table (PRLT) based on the results of the prescheduling. Secondly, to get the priority of the ready task, the actual start time of the ready task is also considered. In this way, the algorithm achieves a balance between the front and back when calculating the priority, and the priority close to the real scheduling is obtained. The experimental results on random application graphs show that the BVSL algorithm performs better than other algorithms on random graphs with a size of 20 to 400 tasks. In the case of high heterogeneity, the algorithm has stronger competition than the RE-18 and IPPTS algorithms. However, as the number of processors continues to increase, the competitiveness of algorithms also gradually declines. The results on the three real-world application graphs show that the overall performance of the BVSL algorithm is also better than the IPPTS and RE-18 algorithms. In the next step, we will study scheduling algorithms in dynamic environments and scheduling algorithms under multiple constraints.

## Data Availability

We have not yet put the result data of experiments to public site of the public network. The result data of experiments can be available from the corresponding author of the manuscript.

## Conflicts of Interest

The authors declare that they have no conflicts of interest.

## Acknowledgments

This work was supported in part by the Youth Program of National Natural Science Foundation of China: "Research on Workflow Scheduling in a Hybrid Cloud Environment" (No.: 62002078), the National Natural Science Foundation of China (Grant No. 62072131), and the Guangdong Province Key Field R&D Program Project: "Human-machine-physical collaborative control and decision-making theory method in complex manufacturing environment" (No.: 2020B0101050001).

## References

[1] S. Lu, R. Gu, H. Jin, L. Wang, X. Li, and J. Li, "QoS-aware task scheduling in cloud-edge environment," *IEEE Access*, vol. 9, pp. 56496–56505, 2021.

[2] S. Chen, Q. Li, M. Zhou, and A. Abusorrah, "Recent advances in collaborative scheduling of computing tasks in an edge computing paradigm," *Sensors*, vol. 21, no. 3, p. 779, 2021.

[3] C. Kai, H. Zhou, Y. Yi, and W. Huang, "Collaborative cloud-edge-end task offloading in mobile-edge computing networks with limited communication capability," *IEEE Transactions on Cognitive Communications and Networking*, vol. 7, no. 2, pp. 624–634, 2021.

[4] H. Yuan and M. Zhou, "Profit-maximized collaborative computation offloading and resource allocation in distributed cloud and edge computing systems," *IEEE Transactions on Automation Science and Engineering*, vol. 18, no. 3, pp. 1277–1287, 2021.

[5] Z. Ning, P. Dong, X. Kong, and F. Xia, "A cooperative partial computation offloading scheme for mobile edge computing enabled Internet of Things," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4804–4814, 2019.

[6] D. Guo, S. Gu, J. Xie, L. Luo, X. Luo, and Y. Chen, "A mobile-assisted edge computing framework for emerging IoT applications," *ACM Transactions on Sensor Networks*, vol. 17, no. 4, pp. 1–24, 2021.

[7] Z. Ning, X. Kong, F. Xia, W. Hou, and X. Wang, "Green and sustainable Cloud of Things: enabling collaborative edge computing," *IEEE Communications Magazine*, vol. 57, no. 1, pp. 72–78, 2019.

[8] A. Singh, K. Chatterjee, and S. C. Satapathy, "An edge based hybrid intrusion detection framework for mobile edge computing," *Complex & Intelligent Systems*, vol. 2021, pp. 1–28, 2021.

[9] J. Zhang, X. Hu, Z. Ning et al., "Joint resource allocation for latency-sensitive services over mobile edge computing networks with caching," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4283–4294, 2019.

[10] R. Gholivand and Z. Movahedi, "A cloud-RAN based end-to-end computation offloading in mobile edge computing," *Computer Communications*, vol. 175, pp. 193–204, 2021.

[11] Z. Ning, X. Wang, J. J. R. C. Rodrigues, and F. Xia, "Joint computation offloading, power allocation, and channel assignment for 5G-enabled traffic management systems," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 5, pp. 3058–3067, 2019.

[12] S. Song, S. Ma, J. Zhao, F. Yang, and L. Zhai, "Cost-efficient multi-service task offloading scheduling for mobile edge computing," *Applied Intelligence*, vol. 2021, pp. 1–13, 2021.

[13] J. Zhang, X. Hu, Z. Ning et al., "Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2633–2645, 2018.

[14] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading tasks with dependency and service caching in mobile edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 11, pp. 2777–2792, 2021.

[15] Z. Ning, P. Dong, M. Wen et al., "5G-enabled UAV-to-community offloading: joint trajectory design and task scheduling," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 11, pp. 3306–3320, 2021.

[16] Z. Ning, S. Sun, X. Wang et al., "Intelligent resource allocation in mobile blockchain for privacy and security transactions: a deep reinforcement learning based approach," *Science China Information Sciences*, vol. 64, no. 6, article 162303, 2021.

[17] Z. Ning, P. Dong, X. Wang et al., "Distributed and dynamic service placement in pervasive edge computing networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 6, pp. 1277–1292, 2021.

[18] X. Wang, Z. Ning, and S. Guo, "Multi-agent imitation learning for pervasive edge computing: a decentralized computation offloading algorithm," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 411–425, 2021.

[19] T. Li, W. Liu, A. Liu et al., "BTS: a blockchain-based trust system to deter malicious data reporting in intelligent Internet of Things," *IEEE Internet of Things Journal*, p. 1, 2021.

[20] T. Li, W. Liu, S. Xie et al., "BPT: a blockchain based privacy information preserving system for trust data collection over distributed mobile edge network," *IEEE Internet of Things Journal*, vol. 2021, p. 1, 2021.

[21] J. Liang, W. Liu, N. Xiong, A. Liu, and S. Zhang, "An intelligent and trust UAV-assisted code dissemination 5G system for industrial Internet-of-Things," *IEEE Transactions on Industrial Informatics*, p. 1, 2021.

[22] W. Mo, W. Liu, G. Huang, N. Xiong, A. Liu, and S. Zhang, "A cloud-assisted reliable trust computing scheme for data collection in Internet of Things," *IEEE Transactions on Industrial Informatics*, vol. 2021, p. 1, 2021.

[23] Z. Ning, S. Sun, X. Wang et al., "Blockchain-enabled intelligent transportation systems: a distributed crowdsensing framework," *IEEE Transactions on Mobile Computing*, vol. PP, p. 1, 2021.

[24] C. Huang, G. Huang, W. Liu, R. Wang, and M. Xie, "A parallel joint optimized relay selection protocol for wake-up radio enabled WSNs," *Physical Communication*, vol. 47, article 101320, 2021.

[25] T. Li, W. Liu, Z. Zeng, and N. N. Xiong, "DRLR:A deep reinforcement learning based recruitment scheme for massive data collections in 6G-based IoT networks," *IEEE Internet of Things Journal*, p. 1, 2021.

[26] F. A. Teixeira, G. V. Machado, F. M. Q. Pereira, H. C. Wong, J. M. S. Nogueira, and L. B. Oliveira, "SIoT: securing the Internet of Things through distributed system analysis," in *14th IEEE/ACM International Conference on Information Processing in Sensor Networks (IPSN)*, pp. 310–321, Seattle, WA, 2015.

[27] J. Lee, "Time-reversibility for real-time scheduling on multiprocessor systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 1, pp. 230–243, 2017.

[28] *Top500, Top 10 sites for JUNE*, 2021, https://www.top500.org/lists/top500/2021/06/.

[29] Y. Zhang, F. Tong, C. Li, and Y. Xu, "Bi-objective workflow scheduling on heterogeneous computing systems using a memetic algorithm," *Electronics*, vol. 10, no. 2, 2021.

[30] N. Kumar and D. P. Vidyarthi, "A novel hybrid PSO-GA meta-heuristic for scheduling of DAG with communication on multiprocessor systems," *Engineering with Computers*, vol. 32, no. 1, pp. 35–47, 2016.

[31] F. Happach, "Makespan minimization with OR-precedence constraints," *Journal of Scheduling*, vol. 24, no. 3, pp. 319–328, 2021.

[32] B. Korte and J. Vygen, "NP-Completeness," in *Combinatorial Optimization*, B. Korte and J. Vygen, Eds., pp. 385–421, Springer, Berlin Heidelberg, 2018.

[33] H. Djigal, J. Feng, and J. Lu, "Task scheduling for heterogeneous computing using a Predict Cost Matrix," in *48th International Conference on Parallel Processing (ICPP)*, pp. 1–10, Kyoto, Japan, 2019.

[34] H. Djigal, J. Feng, J. Lu, and J. Ge, "IPPTS: an efficient algorithm for scientific workflow scheduling in heterogeneous computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 5, pp. 1057–1071, 2021.

[35] 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.

[36] Y. Zhao, S. Cao, L. Yan, and I. C. Soc, "List scheduling algorithm based on pre-scheduling for heterogeneous computing," in *IEEE Int Conf on Parallel and Distributed Processing with Applications, Big Data and Cloud Computing, Sustainable Computing and Communications, Social Computing and Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pp. 588–595, Xiamen, China, 2019.

[37] N. Zhou, D. Qi, X. Wang, Z. Zheng, and W. Lin, "A list scheduling algorithm for heterogeneous systems based on a critical node cost table and pessimistic cost table," *Concurrency and Computation-Practice & Experience*, vol. 29, no. 5, article e3944, 2017.

[38] 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.

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

[40] L. F. Bittencourt, R. Sakellariou, and E. R. M. Madeira, "DAG scheduling using a lookahead variant of the Heterogeneous Earliest Finish Time algorithm," in *18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pp. 27–34, Pisa, Italy, 2010.

[41] H. Gholami and R. Zakerian, "A list-based heuristic algorithm for static task scheduling in heterogeneous distributed computing systems," in *6th International Conference on Web Research (ICWR)*, pp. 21–26, Tehran, Iran, 2020.

[42] K. He, X. Meng, Z. Pan, L. Yuan, and P. Zhou, "A novel task-duplication based clustering algorithm for heterogeneous computing environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 1, pp. 2–14, 2019.

[43] C. Boeres, J. Viterbo, and V. E. F. Rebello, "A cluster-based strategy for scheduling task on heterogeneous processors," in *16th Symposium on Computer Architecture and High Performance Computing*, pp. 214–221, Foz do Iguacu, Brazil, 2004.

[44] H. Chen, X. Zhu, D. Qiu, L. Liu, and Z. Du, "Scheduling for workflows with security-sensitive intermediate data by selective tasks duplication in clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 9, pp. 2674–2688, 2017.

[45] X. Geng, L. Yu, J. Bao, and G. Fu, "A task scheduling algorithm based on priority list and task duplication in cloud computing environment," in *Web Intelligence*, vol. 17, no. 2pp. 121–129, IOS Press, 2019.

[46] R. Madhura, B. L. Elizabeth, and V. R. Uthariaraj, "An improved list-based task scheduling algorithm for fog computing environment," *Computing*, vol. 103, no. 7, pp. 1353–1389, 2021.

[47] Q. Tian, J. Li, D. Xue et al., "A hybrid task scheduling algorithm based on task clustering," *Mobile Networks & Applications*, vol. 25, no. 4, pp. 1518–1527, 2020.

[48] A. Yoosefi and H. R. Naji, "A clustering algorithm for communication-aware scheduling of task graphs on multicore reconfigurable systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 10, pp. 2718–2732, 2017.

[49] M. Karpagam, K. Geetha, and C. Rajan, "A reactive search optimization algorithm for scientific workflow scheduling using clustering techniques," *Journal of Ambient Intelligence*

*and Humanized Computing*, vol. 12, no. 2, pp. 3199–3207, 2021.

[50] H. Wang and O. Sinnen, "List-scheduling versus cluster-scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 8, pp. 1736–1749, 2018.

[51] W. Ahmad and B. Alam, "An efficient list scheduling algorithm with task duplication for scientific big data workflow in," *Heterogeneous Computing Environments*, vol. 33, no. 5, article e5987, 2021.

[52] L. Shi, J. Xu, L. Wang et al., "Multijob associated task scheduling for cloud computing based on task duplication and insertion," *Wireless Communications and Mobile Computing*, vol. 2021, Article ID 6631752, 2021.

[53] K. R. Shetti, S. A. Fahmy, and T. Bretschneider, "Optimization of the HEFT algorithm for a CPU-GPU environment," in *14th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pp. 212–218, Taipei, Taiwan, 2013.

[54] G. B. Berriman, E. Deelman, J. Good et al., "Montage: a grid enabled engine for delivering custom science-grade mosaics on demand," in *Conference on Optimizing Scientific Return for Astronomy through Information Technologies*, vol. 5493, pp. 221–232, Glasgow, Scotland, 2004.

[55] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M. Su, and K. Vahi, "Characterization of scientific workflows," in *3rd Workshop on Workflows in Support of Large-Scale Science*, pp. 1–10, Austin, TX, USA, 2008.

[56] E. Deelman, G. Singh, M.-H. Su et al., "Pegasus: a framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, 237 pages, 2005.