International Journal of Innovative
Computing, Information and Control
Volume **16**, Number **2**, April **2020**

ICIC International ©2020 ISSN 1349-4198
pp. 701–713

PVBTS: A NOVEL TASK SCHEDULING ALGORITHM FOR HETEROGENEOUS COMPUTING PLATFORMS

Chao Jiang[1,2], Jinlin Wang[1,2] and Xiaozhou Ye[1,*]

[1]National Network New Media Engineering Research Center
Institute of Acoustics, Chinese Academy of Sciences
No. 21, North 4th Ring Road, Haidian District, Beijing 100190, P. R. China
{ jiangc; wangjl }@dsp.ac.cn; *Corresponding author: yexz@dsp.ac.cn

[2]School of Electronic, Electrical and Communication Engineering
University of Chinese Academy of Sciences
No. 19(A), Yuquan Road, Shijingshan District, Beijing 100049, P. R. China

Received July 2019; revised November 2019

Abstract. *Efficient task scheduling has always been one of the most critical issues for high performance in heterogeneous computing. The heterogeneity of computation costs on a given set of processors and the communication costs among processors increase the complexity of the scheduling problem. Generally, the application consists of several tasks with dependencies. If the computation costs, task dependencies and communication costs are known a priori, the application can be represented by a static model, namely the directed acyclic graphs (DAG) model. In this paper, we proposed a novel task scheduling algorithm called penalty value based task scheduling (PVBTS) for application scheduling problem. The PVBTS algorithm dynamically determines the execution order of tasks according to the penalty value which is computed based on the heterogeneity of execution completion time on a given set of processors. In each step, the PVBTS algorithm maintains a ready list including all the independent tasks, then selects the task with the highest penalty value and maps it to a processor that gives the minimum execution completion time of the task. The PVBTS algorithm uses randomly generated task graphs and some real-world application task graphs to evaluate performance. The experimental results indicate that the PVBTS algorithm outperforms some well-known scheduling algorithms selected for the performance comparison in terms of schedule length (makespan) and efficiency.*

Keywords: Heterogeneous computing, Task scheduling, Directed acyclic graph, Schedule length, Efficiency

1. **Introduction.** A heterogeneous computing platform is composed of a diverse set of system computing resources, which can be either local or distributed. It is usually used for computationally intensive applications. A heterogeneous multi-core processor is a native heterogeneous computing platform that integrates different types of processors, such as CPU, GPU and FPGA in the same chip. A distributed heterogeneous computing platform typically consists of a diverse set of distributed computing resources interconnected by a high speed network. Cloud computer also follows a distributed and parallel system and resource management and scheduling in cloud computing environment is becoming one of the most complex issues [9]. In order to achieve high performance, the method of scheduling the application tasks on various computing resources in parallel is very important. The scheduling algorithms assign tasks to processors and aim to minimize the execution time with the constraints of task precedence. In the static task scheduling

DOI: 10.24507/ijicic.16.02.701

problem, it is common to use the DAG (directed acyclic graph) model which includes the computation costs of tasks, the communication costs between tasks and the dependencies among tasks to represent the application. The DAG scheduling problem has been proven to be NP-complete [1].

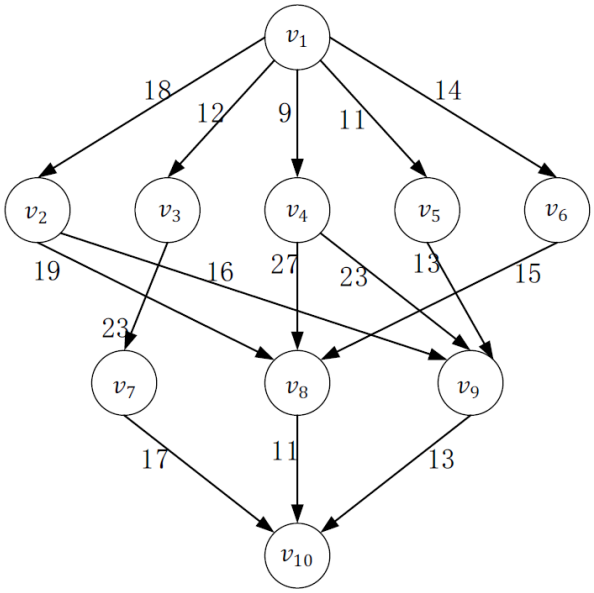
In general, static scheduling algorithms can be classified into two major categories, namely heuristic based and guided random search based algorithms. Heuristic-based group is composed of three subcategories: list-based heuristics, cluster-based scheduling heuristics, and replication-based scheduling heuristics. List-based scheduling heuristics [2-8] typically consist of two phases. In the first phase, task priority is used to generate a ready list of precedence-constrained tasks and the task with the highest priority is selected for scheduling in each step. The second phase assigns tasks to the suitable processors according to the predefined cost criterion. List-based scheduling heuristics can provide a better performance with a lower time complexity than the other heuristics. Cluster-based scheduling heuristics [10-15] are usually used for an unlimited number of processors. A clustering heuristic requires additional steps to merge the task clusters generated by the algorithm onto an unlimited number of processors and sort the execution order of tasks within each processor. The main idea of the replication-based scheduling heuristics [16-21] is to use the free time of the processor to duplicate the predecessor task, which can decrease the inter-process communication overhead. However, it has a higher time complexity than others, which leads to a lower efficiency. The guided random search based algorithms mainly include genetic algorithms [24-27], which may have lower scheduling length, but tend to have higher complexity.

In this paper, we propose a novel heterogeneous list scheduling heuristic called penalty value based task scheduling (PVBTS) to solve static task scheduling problems for a bounded number of fully connected heterogeneous processors. The remainder of this paper is organized as follows. Section 2 gives the static task scheduling problem statement and defines some parameters utilized in the algorithm. Section 3 introduces related work in DAG scheduling and gives a brief overview of the scheduling algorithms selected for performance comparison with the proposed PVBTS algorithm. Section 4 presents the PVBTS algorithm in detail. Section 5 evaluates the performance of the algorithms based on a large number of randomly generated task graphs and some real-world application graphs. Section 6 concludes this paper.

2. **Task-Scheduling Problem.** In this paper, we consider a heterogeneous multi-core platform with q heterogeneous computational processors, $P = \{p_1, p_2, \ldots, p_q\}$. The speeds of the processors in the set are different from each other. It is assumed that all processors are completely inter-connected without contention between them. In addition, we also assume that computation can overlap with communication.

In the general form of static task scheduling, the application is represented by the directed acyclic graph (DAG), $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ denotes the set of application tasks and E denotes the set of edges that represents the precedence constraint between tasks. Task v_j cannot start execution until task v_i completes its execution in case that v_i is the parent task of v_j. The data exchanges between tasks are represented by an $n \times n$ matrix which is denoted by $Data$, where $data_{i,j}$ represents the amount of transmitted data from the task v_i to the task v_j. The computation cost matrix is represented by an $n \times q$ matrix W, where $w_{i,m}$ gives the estimated time to execute task v_i on processor p_m. An example application graph with 10 nodes and the computation cost matrix are presented in Figure 1.

The data transfer rates between any two processors are represented by a $q \times q$ matrix B, where $B_{m,n}$ gives the data transfer rate between processor p_m and processor p_n. The



Computation Cost Matrix

Task	p_1	p_2	p_3
1	14	16	9
2	13	19	18
3	11	13	19
4	13	8	17
5	12	13	10
6	13	16	9
7	7	15	11
8	5	11	14
9	18	12	20
10	21	7	16

FIGURE 1. An example task graph and computation time matrix

communication startup costs of processors are represented by a q-dimensional vector S, where S_m gives the communication startup time of processor p_m. The communication cost of transferring data from task v_i (scheduled on processor p_m) to task v_j (scheduled on processor p_n) is defined as:

$$c_{i,j} = S_m + \frac{data_{i,j}}{B_{m,n}}, \tag{1}$$

when two tasks v_i and v_j are scheduled on the same processor, $c_{i,j}$ becomes zero because it is negligible compared with interprocessor communication costs.

Let $EST(v_i, p_m)$ and $EFT(v_i, p_m)$ denote the earliest start time (EST) and earliest finish time (EFT) of task v_i on processor p_m, respectively. For the entry task v_{entry}, $EST(v_{entry}, p_m) = 0$. The EFT and EST of other tasks in DAG can be computed recursively by traversing the DAG downward starting from the entry node v_{entry} using Equation (2) and Equation (3), respectively.

$$EST(v_i, p_m) = \max\left\{avail[m], \max_{v_j \in pred(v_i)} \left(AFT(v_j) + c_{j,i}\right)\right\}, \tag{2}$$

$$EFT(v_i, p_m) = EST(v_i, p_m) + w_{i,m}, \tag{3}$$

where $pred(v_i)$ is the set of immediate predecessor tasks of task v_i and $avail[m]$ is the earliest time at which processor p_m is ready for task execution. The inner max block in Equation (2) represents the time when all data needed by task v_i has arrived at processor p_m.

The schedule length (makespan) also termed as makespan of the task graph denotes the execution completion time of the exit task. It is defined as:

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

The minimum earliest finish time (mEFT) of a task v_i is the minimum value of the EFT values of the task on a set of processors. The mEFT is defined as:

$$mEFT(v_i) = \min_{p_n \in P}\{EFT(v_i, p_n)\}. \tag{5}$$

The penalty value (PV) of a task v_i represents the heterogeneity of execution completion time on the processors. The penalty value is based on the concept that a task will be penalized when it misses the opportunity to execute on the processor that could execute it faster than all the other processors. Therefore, we assign a penalty value to each task as the priority. A task with a higher PV value may increase the schedule length if not scheduled with a higher priority. The PV of a task is defined as:

$$PV(v_i) = \frac{\sum_{m=1}^{q} EFT(v_i, p_m)}{mEFT(v_i)}. \tag{6}$$

3. **Related Work.** A plethora of task scheduling algorithms have been proposed for static application scheduling problems in heterogeneous computing platform. It is popular to use list-based scheduling algorithm to solve static task scheduling problems. Many list scheduling algorithms have been proposed such as mapping heuristic (MH) [3], dynamic level scheduling (DLS) [5], heterogeneous earliest finish time (HEFT) [2], critical path on a processor (CPOP) [2], performance effective task scheduling (PETS) [4], predicted earliest finish time (PEFT) [8] and SD-based algorithm for task scheduling (SDBATS) [6], heterogeneous scheduling algorithm with improved task priority (HSIP) [7].

Here, we provide a brief overview of some most cited list-based scheduling algorithms selected for the performance comparison with the proposed PVBTS algorithm, namely HEFT, PETS and PEFT.

3.1. **Heterogeneous earliest finish time.** The HEFT algorithm first calculates the upward rank called $rank_u$ for each task based on the average computation and average communication costs. The $rank_u$ of task v_i represents the length of the longest path (critical path) from task v_i to the exit task, including the computational cost of task v_i. The ready list includes all the tasks sorted in descending order of $rank_u$. Then the HEFT algorithm calculates the earliest finish time (EFT) of each task in the processor selection phase. Meanwhile, the insertion-based policy is employed to take full advantage of the idle time slots if there exists. The selected task will be scheduled on the processor which gives the minimum EFT of the task. For a given DAG with v tasks and q heterogeneous processors, the time complexity of the HEFT algorithm is $O\big(v^2 \times q\big)$.

3.2. **Performance effective task scheduling.** The PETS algorithm first groups tasks according to task level so that tasks in the same group can be executed in parallel. The execution order of tasks in the same group is determined by the priority of each task which is calculated based on the communication costs and the average computation cost of task. The communication costs include the cost of receiving data from its predecessor task (DRC) and transmitting data to all its successor tasks (DTC). Then the PETS algorithm uses the same strategy as the HEFT algorithm for processor selection phase. The time complexity of the PETS algorithm is $O\big(v^2 \times (q \times \log v)\big)$.

3.3. **Predicted earliest finish time.** The PEFT algorithm is based on optimistic cost table (OCT) which is used for calculating task priority and mapping tasks to processors. Task priority is determined by computing the average OCT on the given set of processors. In processor selection phase, the optimistic earliest finish time (OEFT) which represents the sum of the optimistic cost and earliest finish time is used instead of the earliest finish time (EFT). Then the task is scheduled on the processor that gives the minimum optimistic earliest finish time (OEFT). The time complexity of PEFT algorithm is $O\big(v^2 \times q\big)$.

Because of its good scheduling quality and low time complexity, HEFT algorithm has become the most popular and widely used list scheduling algorithm. PETS algorithm

improves the calculation method of task priority by taking account of both data transfer cost (DTC) and data receiving cost (DRC), so it has better scheduling quality than HEFT algorithm. PEFT algorithm introduces the look ahead feature while maintaining the same time complexity as HEFT algorithm. PEFT algorithm outperforms other list scheduling algorithms such as HEFT and PETS in terms of scheduling length and efficiency. However, the performance improvement of PEFT algorithm decreases as the size of task graph increases, because the uncertainty of subtasks increases greatly for larger DAGs.

A large number of list scheduling algorithms calculate the priorities of tasks based on the average computation cost on a given set of processors and the average communication cost, which leads to a lack of consideration of the heterogeneity of the computation cost. In addition, the task priority will not change even if a failure occurs at a particular processor, which means the computing resource state at runtime is not taken into account. To overcome these limitations, we proposed the PVBTS algorithm.

4. **Proposed Algorithm.** In this section, we present the proposed PVBTS algorithm in detail. The PVBTS algorithm can be divided into two phases, one is the task prioritizing phase and the other is the processor selection phase. These two phases are implemented alternately. In the task prioritizing phase, we construct an independent task list and dynamically determine the execution order of the tasks based on the predefined criterion, namely the penalty value. Then we map the task with the highest PV value to the best processor which minimizes the earliest finish time in the processor selection phase. Each time a task is assigned to a processor, the ready list is updated and the priority of tasks in the list is recalculated. The partial task duplication policy is employed to reduce the schedule length.

4.1. **Task prioritization phase.** Some list scheduling algorithms group tasks according to task levels. As a result, tasks are independent of each other at each level and can be executed in parallel. However, this approach does not take account of the change of the constraints of task precedence after mapping a task to the best processor. The proposed algorithm constructs a dynamic ready task list initialized with the entry task. An independent task refers to a task which has all its predecessor tasks finished execution. All the independent tasks are inserted into the ready task list. The penalty value (PV) is used as the prioritization criterion to determine the order of executing the tasks in the ready list. The PV value of a task represents the heterogeneity of execution completion time on the given set of processors. It is calculated using Equation (6). After a task is assigned to a processor, it will be removed from the ready list. Then we check if a new independent task is produced. If such a task exists, it will be added to the ready list. The PV value of tasks in the ready list will be recalculated. This process continues until the ready list becomes empty which represents all the tasks have been scheduled.

4.2. **Processor selection phase.** In the processor selection phase, the input is the ready task list including all the current independent tasks. The task with the highest PV value is selected and mapped to the processor which gives the minimum earliest finish time (EFT). Then the selected task will be removed from the ready list.

To avoid the overhead caused by the task duplication, only the duplication of entry task is taken into account. The entry task follows the duplication policy.

1) Choose the processor p_m which gives the minimum earliest finish time (EFT) for the entry task v_{entry};
2) Determine whether the entry task v_{entry} needs to be duplicated on a particular processor p_n according to the following condition. For each task $v_i \in succ(v_{entry})$, if Equation

(7) is satisfied, then entry task duplication is performed on processor p_n. Otherwise, there is nothing to do.

$$w_{entry,n} < w_{entry,m} + c_{entry,i}, \tag{7}$$

where $succ(v_{entry})$ is the set of immediate successor tasks of task v_{entry} and $c_{entry,i}$ is the communication cost between the entry task v_{entry} and its successor task v_i.

4.3. **Detailed description of the PVBTS algorithm.** In this section, we give the description of each step of the PVBTS algorithm in detail through an example. Algorithm 1 shows the pseudo-code of the PVBTS algorithm.

The proposed PVBTS algorithm dynamically assigns priority to the tasks at each step and also takes account of the state of computing resources at runtime. So the PVBTS algorithm has better load balancing. The PVBTS algorithm can effectively tolerate malfunctions of the processors in the heterogeneous computing platform and improves system reliability. The complexity of PVBTS algorithm is $O\big(v^2 \times (\log v \times q)\big)$ in terms of the number of tasks v and the number of processors q.

Algorithm 1. The PVBTS Algorithm

Input: A DAG $G = (V, E)$, set of tasks V, set of processors P
Output: Schedule result, makespan

```
1.  Construct a ready list L and initialize with entry task.
2.  while L is not empty do
3.      for each task v_i in L do
4.          for each processor p_m ∈ P do
5.              Compute the EST(v_i, p_m) and EFT(v_i, p_m) values using
                Equation (2) and Equation (3), respectively.
6.          end
7.          Compute the PV(v_i) value using Equation (6).
8.      end
9.  Remove the task v_i with the highest PV value from L.
10. if task v_i is the entry task then
11.     Use the entry task duplication policy as described in Section 4.2.
12. else
13.     assign it to the processor p_m which minimizes EFT(v_i, p_m).
14. end if
15. Update the independent tasks ready list L with the successors of task v_i.
16. end while
```

For the given DAG in Figure 1, Table 1 presents the steps of PVBTS algorithm. Initially, only the entry task T1 is added to the independent task ready list L. The penalty value of T1 is 4.33 and the execution finish time on processors P1, P2 and P3 is 14, 16 and 9, respectively. Therefore, task T1 will be assigned to processor P3. After that, the tasks T2, T3, T4, T5 and T6 become independent and will be inserted into the ready list L. Then we calculate the penalty value and EFT value for each task in L and remove the task T6 which has the highest PV value 4.28 from L. The task T6 will be assigned to P3 according to the EFT value. Here, the entry task duplication policy is checked when calculating the EFT value. At each step, we check whether new tasks not in the list L become independent after a task in L is scheduled. If there exist such tasks, the ready list L is updated with these tasks. The above process will be repeated until all tasks in DAG have been scheduled. The schedule length is calculated using Equation (5). For the

TABLE 1. Schedule produced by the PVBTS algorithm at each step

Step	Ready Task List	Penalty Values	Task Selected	EFT			Processor Selected
				P1	P2	P3	
1	v_1	4.33	v_1	14	16	9	P3
2	v_2, v_3, v_4, v_5, v_6	3.30, 3.28, 3.21, 3.89, 4.28	v_6	27	32	18	P3
3	v_2, v_3, v_4, v_5	3.63, 3.64, 3.58, 3.19	v_3	25	29	37	P1
4	v_2, v_4, v_5, v_7	3.11, 4.04, 3.36, 4.81	v_7	32	63	59	P1
5	v_2, v_4, v_5	3.31, 4.33, 3.61	v_4	45	24	35	P2
6	v_2, v_5	3.44, 3.89	v_5	44	37	28	P3
7	v_2	3.12	v_2	45	43	46	P2
8	v_8, v_9	3.65, 3.84	v_9	77	55	79	P2
9	v_8	3.17	v_8	67	66	76	P2
10	v_{10}	3.62	v_{10}	98	73	93	P2

given DAG in Figure 1, the schedule lengths of the PVBTS, HEFT, PETS and PEFT are 73, 80, 77 and 86, respectively.

5. **Experimental Results.** This section presents a performance evaluation of the proposed PVBTS in comparison with the HEFT, PETS and PEFT algorithms. For this purpose, we make use of two kinds of graph sets, namely randomly generated application graphs and some real-world applications graphs. First of all, we give the comparison metrics for performance evaluation.

5.1. **Comparison metrics.** The performance evaluation of the algorithms is based on the following comparison metrics.

1) Schedule length ratio (SLR)

The metric most commonly used to evaluate the performance of a schedule algorithm on a single DAG is the schedule length of the schedule result. Since a large set of task graphs with different properties is used, it is necessary to normalize the schedule length to a lower bound, which is called the schedule length ratio (SLR) [2]. The SLR is defined as follows:

$$SLR = \frac{makespan}{\sum_{v_i \in CP_{\min}} \min_{p_m \in P}\{w_{i,m}\}}, \tag{8}$$

where $CP_{\min}$ represents the critical path of the given DAG.

2) Speedup

This speedup value is defined as the ratio of the sequential execution time to the parallel execution time [2]. The sequential execution time is the smallest value of the execution time obtained by executing all the tasks in DAG on one processor. The parallel execution time, also known as scheduled length or makespan, is the execution completion time of the application graph. The speedup value is computed as follows:

$$Speedup = \frac{\min_{p_m \in P}\left\{\sum_{v_i \in V} w_{i,m}\right\}}{makespan}. \tag{9}$$

3) Efficiency

The efficiency is defined as the ratio of the speedup value to the number of processors used for scheduling the task graph.

$$Efficiency = \frac{Speedup}{Number\ of\ Processors}. \tag{10}$$

5.2. **Randomly generated task graph.** For the performance evaluation, we first implement a random graph generator to generate application DAGs with the same parameters for synthetic DAG generation as described in [2,8]. We create a pseudo entry (exit) task with zero computation cost and communication cost if there is more than one entry (exit) task in the randomly generated application graph. The following parameters determine the different characteristics of the randomly generated task graphs and are defined as:

- n: Number of application tasks in the DAG;
- α: This parameter affects the shape of the DAG. The height of a DAG is randomly selected from a uniform distribution with the mean value equal to $\sqrt{n}/\alpha$ and the width of each level is randomly selected from a uniform distribution with mean value equal to $\sqrt{n} \times \alpha$ [8]. Therefore, a high value of α leads to a fat DAG with a high parallelism between tasks;
- density: This parameter determines the out-degree of tasks, which means a higher value leading to more edges;
- β: This parameter represents the heterogeneity factor, which affects the range percentage of computation costs on processors. A higher value of β causes a higher variation in a task's computation costs among the processors. The average computation time of each task v_i is selected randomly from a uniform distribution with range $[0,\ 2 \times \overline{w_{DAG}}]$ [2]. Then the computation time of each task v_i on each processor p_m is randomly set using the following relationship:

$$\overline{w_i} \times \left(1 - \frac{\beta}{2}\right) \leq w_{i,m} \leq \overline{w_i} \times \left(1 + \frac{\beta}{2}\right). \tag{11}$$

- CCR: Communication to computation ratio, which is defined as the ratio of the sum of the edge weights to the sum of the node weights in a DAG. A low value of CCR represents a computation-intensive application;
- p: This parameter represents the number of processors used for scheduling the task graphs.

In our experiment, the ranges of values for these parameters are given in the following sets.

- $n = [20, 50, 100, 200, 400]$,
- $\alpha = [0.2, 0.5, 1, 1.5, 2]$,
- $density = [1, 2, 3, 4, 5]$,
- $\beta = [0.2, 0.5, 1, 1.5, 2]$,
- $CCR = [1, 2, 3, 4, 5]$,
- $p = [2, 4, 6, 8, 10]$.

These combinations can generate 15625 unique DAGs. We generate 20 different random graphs with various node and edge weights for each combination of the parameters. Thus, 312500 random graphs are used in the experiment. In our experiment, n, CCR, and p are selected for performance comparison. The other parameters are used for generating task graphs with various characteristics to illustrate the results statistically. For the comparison between PVBTS and selected scheduling algorithms, we calculate the average value of the performance metrics defined in Section 4.1 in terms of the selected parameter.

Figure 2(a) and Figure 2(b) show the average SLR for all algorithms as a function of the DAG size and CCR value, respectively. The results show that PVBTS and HEFT present similar results when the CCR is of a low value. However, with the increasing of CCR value, the PVBTS algorithm outperforms the other selected scheduling algorithms used in the evaluation. In other words, the PVBTS algorithm is more suitable for communication intensive applications. As for the task graph size, the PVBTS outperforms better for

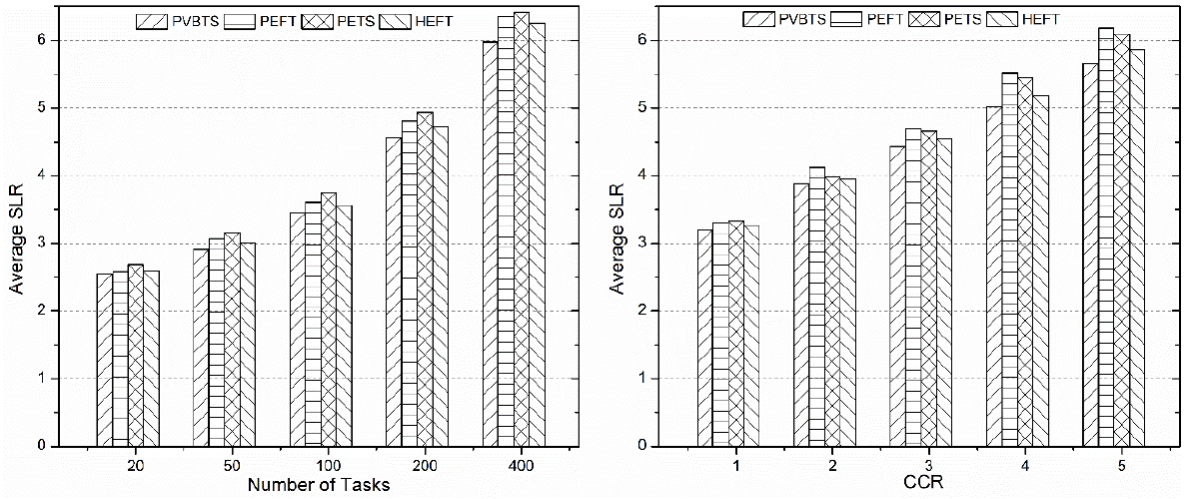


FIGURE 2. Average SLR for random graphs as a function of (a) DAG size and (b) CCR value

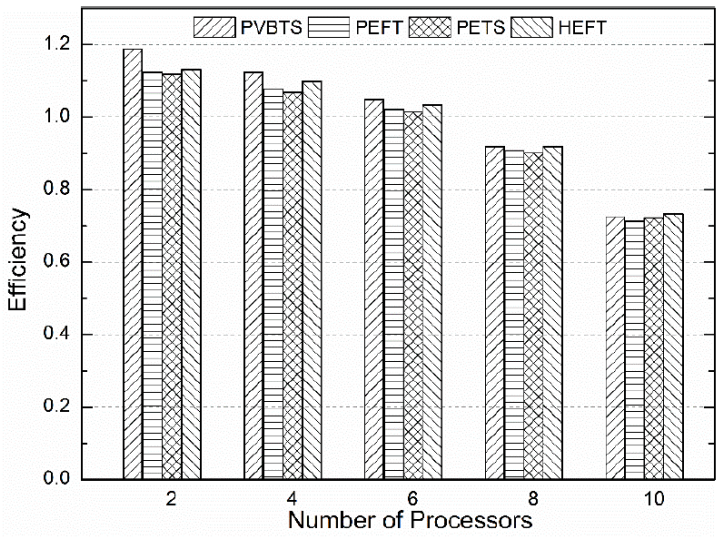


FIGURE 3. Efficiency for random graphs as a function of number of processors

larger DAGs. Figure 3 shows the comparison result of the efficiency as a function of the number of processors. We can find out that the PVBTS performs much better than the other list scheduling algorithms for a small number of processors, but efficiency of PVBTS will decrease as the number of processors increases. This is because PVBTS only considers all the independent tasks in each step, regardless of the overall structure of the task graph and the impact of the processor selection of the task on its child tasks.

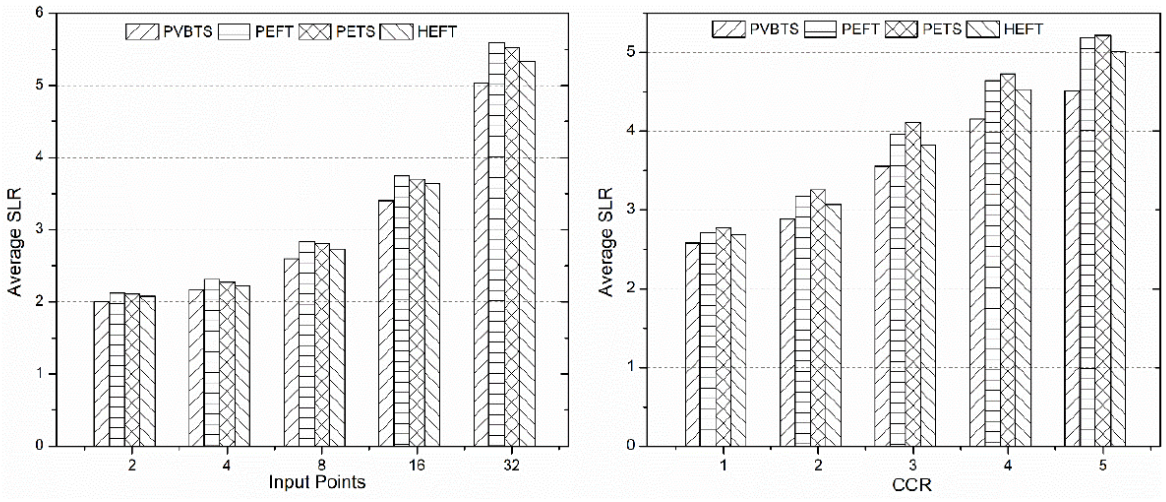
The experiment results show that PVBTS has the lowest average SLR and has better performance improvement for task graphs with higher CCR value or larger DAG size. That is to say, the proposed PVBTS algorithm is more suitable for task graph with complex structure. This is because PVBTS dynamically determines the execution order of tasks based on the heterogeneity of computation cost. In addition, we consider the real-time state of computing resources at each step of the PVBTS algorithm so that we can deal with uncertain situations better and tolerate processor failures.

5.3. **Real-world application graphs.** In order to prove the usefulness of the proposed algorithm, a few of real-world applications such as, Fast Fourier Transform [2] and Montage [22,23], are used for the performance evaluation of the algorithms. Since the structure of these applications is known, we simply use various values for parameters such as CCR and heterogeneity factor.

5.3.1. *Fast Fourier transform.* The FFT task graph is composed of two parts: the recursive call tasks and butterfly operation tasks. For an FFT application graph with N points, the total number of recursive call tasks and butterfly operation tasks are equal to $2 \times (N-1)+1$ and $N \times \log_2 N$, respectively.

In our experiments, we change the value of input points from 2 to 32 with an increment of powers of 2, which generates task graphs with various size from the smallest size 5 to the largest size 223. Because of known structure of FFT task graph, only the CCR, β, and number of processors parameters are considered for our evaluation.

The comparison results of the average SLR against the FFT input points and CCR values are shown in Figure 4(a), and Figure 4(b), respectively. For the evaluation of efficiency, we set the size input points to 16 and change the number of processors from 2 to 10. Figure 5 shows the result that the efficiency improvement of the PVBTS algorithm decreases with the increasing number of processors. It is clear that the PVBTS algorithm reveals performance improvement over all the other selected algorithms in terms of average SLR and efficiency. We can conclude that PVBTS exhibits better performance improvement for task graphs with the same characteristics as FFT and that are characterized by having all tasks belonging to a critical path.



(a) (b)

FIGURE 4. Average SLR for FFT application graphs as a function of (a) input points and (b) CCR value

5.3.2. *Montage.* The Montage is used for generating astronomical image mosaics of the sky. Because the structure of Montage task graph is fixed as well, we use the same parameters as FFT task graph to generate task graphs for performance evaluation, namely the CCR value, heterogeneous factor β and number of processors. In our evaluation, we consider Montage task graphs with 50 and 100 nodes. First, we set the number of processors to 4. For each value of the CCR in the range between 1 to 5, the average SLR values obtained by PVBTS, PEFT, PETS and HEFT algorithms are shown in Figure 6. To evaluate the efficiency, we keep the CCR value at 3 and change the number of processors

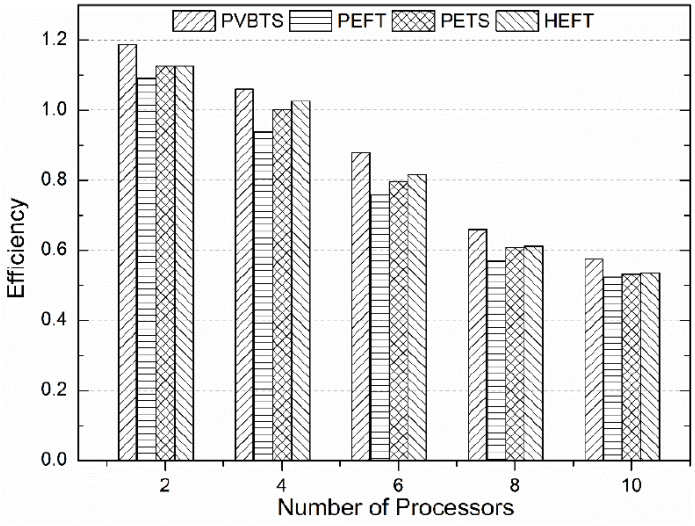


FIGURE 5. Efficiency for FFT application graphs as a function of number of processors

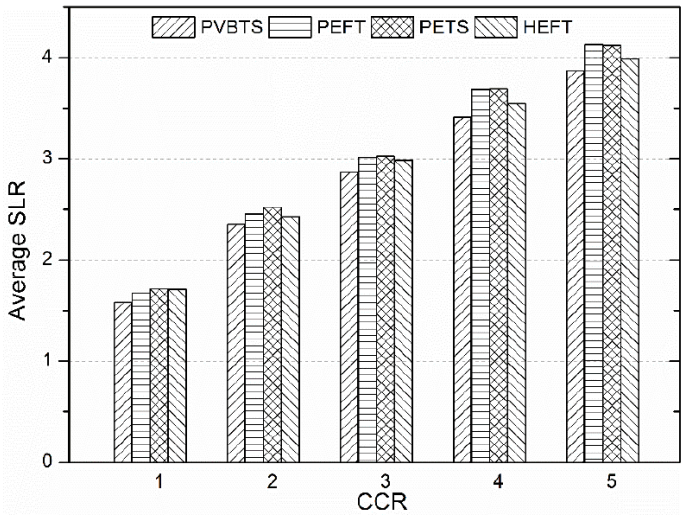


FIGURE 6. Average SLR for Montage application graphs as a function of CCR value

from 2 to 10. Figure 7 shows the efficiency comparison result for various processor number. The results show that for real-world Montage application graphs, PVBTS outperforms the PEFT, PETS and HEFT algorithms in terms of the average SLR and efficiency.

6. **Conclusions.** In order to take full advantage of the high performance of heterogeneous computing platforms, efficient and robust scheduling algorithms are necessary. A novel task scheduling algorithm called penalty value based task scheduling (PVBTS) is proposed in this paper. The scheduling list of the PVBTS algorithm is not generated statically according to the predefined priority criterion. At each step of the algorithm, the task ready list contains all independent tasks. For each task in the list, the penalty value of the task is calculated based on the heterogeneity of the execution completion time of the task on the diverse set of processors and the task with the highest penalty value is selected and scheduled first. The proposed algorithm considers the state of computing resources when calculating priority and assigning tasks to processors, so it performs well in load balancing. Besides, the entry task duplication policy is used to reduce the scheduling

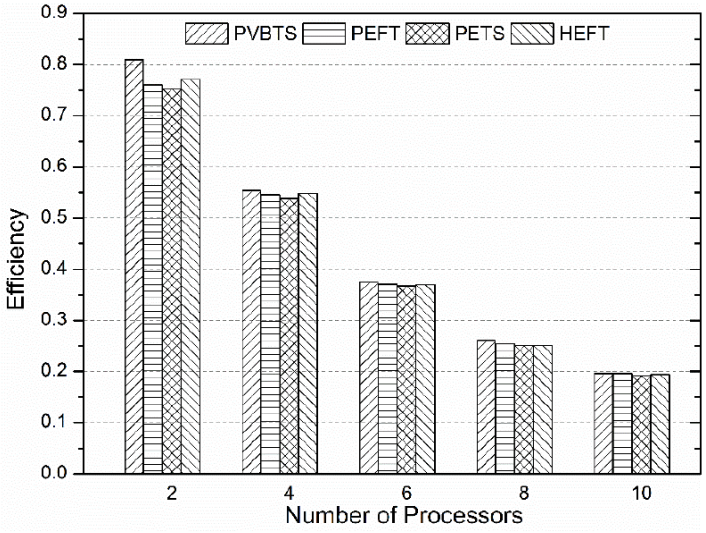


FIGURE 7. Efficiency for Montage application graphs as a function of number of processors

length. Since the PVBTS algorithm dynamically assigns priority to tasks, it can be more efficient for uncertain situations in a heterogeneous computing platform. The complexity of the PVBTS algorithm for mapping v tasks to q processors is $O\big(v^2 \times (\log v \times q)\big)$.

The performance comparison is based on a large set of randomly generated task graphs with various characteristics and several task graphs of real-world applications, such as Fast Fourier Transform and Montage. The comparative study reveals that the PVBTS algorithm is an efficient list scheduling algorithm, which shows performance improvement for DAG scheduling in terms of scheduling length and efficiency compared with other existing algorithms such as HEFT, PETS and PEFT.

Acknowledgment. This work is partially supported by Strategic Leadership Project of Chinese Academy of Sciences: SEANET Technology Standardization Research System Development (Project No. XDC02010701).

REFERENCES

[1] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 1979.

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

[3] T. D. Braun, H. J. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys and B. Yao, A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems, *Proc. of the 17th IEEE Symposium on Reliable Distributed Systems*, pp.330-335, 1998.

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

[5] G. C. Sih and E. A. Lee, A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures, *IEEE Transactions on Parallel and Distributed Systems*, vol.4, no.2, pp.75-87, 1993.

[6] E. U. Munir, S. Mohsin, A. Hussain, M. W. Nisar and S. Ali, SDBATS: A novel algorithm for task scheduling in heterogeneous computing systems, *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and PhD Forum*, Cambridge, MA, pp.43-53, 2013.

[7] G. Wang, H. Guo and Y. Wang, A novel heterogeneous scheduling algorithm with improved task priority, *2015 IEEE the 17th International Conference on High Performance Computing and Communications, 2015 IEEE the 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE the 12th International Conference on Embedded Software and Systems*, New York, NY, pp.1826-1831, 2015.

[8] 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.

[9] M. Al Rawajbeh, Performance evaluation of a computer network in a cloud computing environment, *ICIC Express Letters*, vol.13, no.8, pp.719-727, 2019.

[10] A. Niyom and P. Sophatsathit, An energy-efficient process clustering assignment algorithm for distributed system, *Simulation Modelling Practice and Theory*, pp.95-111, 2014.

[11] U. Boregowda et al., A hybrid task scheduler for DAG applications on a cluster of processors, *2014 the 4th ICACC*, pp.143-146, 2014.

[12] T. Yang and A. Gerasoulis, DSC: Scheduling parallel tasks on an unbounded number of processors, *IEEE Transactions on Parallel and Distributed Systems*, vol.5, no.9, pp.951-967, 1994.

[13] B. Cirou and E. Jeannot, Triplet: A clustering scheduling algorithm for heterogeneous systems, *The 30th International Workshops on Parallel Processing (ICPP 2001 Workshops)*, pp.231-236, 2001.

[14] J. Liou and M. A. Palis, An efficient clustering heuristic for scheduling DAGs on multiprocessors, *Proc. of Symp. Parallel and Distributed Processing*, 1996.

[15] D. Bozdag, U. Catalyurek and F. Ozguner, A task duplication based bottom-up scheduling algorithm for heterogeneous environments, *Proc. of the 20th International Parallel and Distributed Processing Symposium 2006 (IPDPS 2006)*, 2006.

[16] Y.-C. Lee and A. Zomaya, A novel state transition method for metaheuristic-based scheduling in heterogeneous computing systems, *IEEE Transactions on Parallel and Distributed Systems*, vol.19, no.9, pp.1215-1223, 2008.

[17] S. Darbha and D. P. Agrawal, Optimal scheduling algorithm for distributed-memory machines, *IEEE Transactions on Parallel and Distributed Systems*, vol.9, no.1 pp.87-95, 1998.

[18] M. Kun and C. Ch, An adaptive scheduling algorithm for scheduling tasks in computational grid, *Proc. of the 2008 the 7th International Conference on Grid and Cooperative Computing*, pp.24-26, 2008.

[19] I. Ahmad and Y. Kwok, A new approach to scheduling parallel programs using task duplication, *Proc. of Int'l Conf. Parallel Processing*, vol.2, pp.47-51, 1994.

[20] B. Kruatrachue and T. G. Lewis, Grain size determination for parallel processing, *IEEE Software*, pp.23-32, 1988.

[21] Y. Chung and S. Ranka, Applications and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed memory multiprocessors, *Proc. of Supercomputing*, pp.512-521, 1992.

[22] G. B. Berriman, J. C. Good, A. C. Laity, A. Bergou, J. Jacob, D. S. Katz, E. Deelman, C. Kesselman, G. Singh, M.-H. Su and R. Williams, Montage: A grid enabled image mosaic service for the national virtual observatory, *Proc. of Astronomical Data Analysis Software and Systems (ADASS) XIII*, pp.593-596, 2004.

[23] E. Deelman, G. Singh, M. H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob and D. S. Katz, Pegasus: A framework for mapping complex scientific workflows onto distributed systems, *J. Scientific Programming*, vol.13, no.3, pp.219-237, 2005.

[24] S. G. Ahmad, C. S. Liew, E. U. Munir et al., A hybrid genetic algorithm for optimization of scheduling workflow applications in heterogeneous computing systems, *Journal of Parallel and Distributed Computing*, vol.87, pp.80-90, 2015.

[25] C. Yang and B. Yang, Parallelization of genetic algorithm based on cilk technology, *Journal of Network New Media*, no.5, pp.54-60, 2012.

[26] M. Akbari, H. Rashidi and S. H. Alizadeh, An enhanced genetic algorithm with new operators for task scheduling in heterogeneous computing systems, *Engineering Applications of Artificial Intelligence*, vol.61, pp.35-46, 2017.

[27] Y. Xu, K. Li, T. T. Khac and M. Qiu, A multiple priority queueing genetic algorithm for task scheduling on heterogeneous computing systems, *2012 IEEE the 14th International Conference on High Performance Computing and Communication & 2012 IEEE the 9th International Conference on Embedded Software and Systems*, Liverpool, pp.639-646, 2012.