Egyptian Informatics Journal (2012) **13**, 135–145

Cairo University

Egyptian Informatics Journal

www.elsevier.com/locate/eij
www.sciencedirect.com

ORIGINAL ARTICLE

Dynamic task scheduling algorithm with load balancing for heterogeneous computing system

Doaa M. Abdelkader *, Fatma Omara

Faculty of Computers and Information, Cairo University, Egypt

Received 18 October 2011; revised 12 March 2012; accepted 4 April 2012
Available online 4 July 2012

KEYWORDS

Makespane;
Heterogeneous system;
Load balance;
Sleek time

Abstract In parallel computation, the scheduling and mapping tasks is considered the most critical problem which needs High Performance Computing (HPC) to solve it by breaking the problem into subtasks and working on those subtasks at the same time. The application sub tasks are assigned to underline machines and ordered for execution according to its proceeding to grantee efficient use of available resources such as minimize execution time and satisfy load balance between processors of the underline machine. The underline infrastructure may be homogeneous or heterogeneous. Homogeneous infrastructure could use the same machines power and performance. While heterogeneous infrastructure include machines differ in its performance, speed, and interconnection. According to work in this paper a new dynamic task scheduling algorithm for Heterogeneous called a Clustering Based HEFT with Duplication (CBHD) have been developed. The CBHD algorithm is considered an amalgamation between the most two important task scheduling in Heterogeneous machine, The Heterogeneous Earliest Finish Time (HEFT) and the Triplet Clustering algorithms. In the CBHD algorithm the duplication is required to improve the performance of algorithm. A comparative study among the developed CBHD, the HEFT, and the Triplet Cluster algorithms has been done. According to the comparative results, it is found that the developed CBHD

* Corresponding author.
E-mail addresses: Mohammed_doaa@hotmail.com (D.M. Abdelkader), F.Omara@Fci-cu.edu.eg (F. Omara).

Peer review under responsibility of Faculty of Computers and Information, Cairo University.
http://dx.doi.org/10.1016/j.eij.2012.04.001

Production and hosting by Elsevier

algorithm satisfies better execution time than both HEFT algorithm and Triplet Cluster algorithm, and in the same time, it achieves the load balancing which considered one of the main performance factors in the dynamic environment.

1. Introduction

The problem of task mapping in heterogeneous systems is finding proper assignment of tasks to processors in order to optimize some performance metric such as the system utilization, load balancing and the minimum execution time [1,2]. Therefore, scheduling algorithm is needed to overcome this restriction. According to the task scheduling problem, the application is represented as a Directed Acyclic Graph (DAG) where nodes (or tasks) represent the needed computation and edges represent the communication between tasks. For each node in the DAG, a weight is assigned corresponding to computation cost, and weights for edges are assigned corresponding to communication cost between nodes [3]. On the other hand, the task scheduling can be either static or dynamic. In the static scheduling algorithms, the decision is made prior the execution time when the resources requirement estimated, while the dynamic scheduling algorithms allocate/reallocate resources at run time [3,4]. The comparative study among different task scheduling algorithms are introduced [2,4]. The most important dynamic algorithms are HEFT, Clustering, and Genetic algorithms [3,5,6].

The work in this paper concerns the dynamic scheduling for Heterogeneous Computing System (HC). An algorithm has been developed to produce efficient task scheduling and mapping for tasks on heterogeneous machines which is called Clustering Based HEFT with Duplication (CBHD). The developed CBHD algorithm is considered an amalgamation between the Triplet Clustering algorithm, by clustering tasks into interrelated group, and the HEFT algorithm by order tasks in each cluster according to rank value, and some tasks may be duplicated on processors in order to improve the performance of scheduling algorithm specially load balance [7], and in the same time, overcome computation overhead.

The paper is organized as follows; the HEFT and the Triplet Clustering algorithms will be introduced in more details in Sections 2 and 3 respectively. The developed CBHD algorithm has been introduced in Section 4. In Section 5, the performance evaluation and comparative study among the developed CBHD, HEFT, and Triplet Cluster algorithms will be introduced. The conclusion is introduced in Section 6.

2. The Heterogeneous Earliest Finish Time (HEFT) algorithm

Among the scheduling algorithms for heterogeneous system, the Heterogeneous Earliest Finish Time (HEFT) algorithm has been shown to produce shorter schedule lengths more often than comparable algorithms [3]. The main concepts of the HEFT algorithm is that the application subtasks are scheduled based on a value computing by what is called a **ranking** function, this function is based on the weights assigned to the subtasks, as well as, the communication between this subtasks [3,8].

The HEFT algorithm works as follows; first, a weight is assigned to each node and edge of the graph, based on the **average** computation and communication respectively which is represented in equation [1]. Then, the graph is traversed upwards and a rank value is assigned to each node. the **ranking** function of the node is calculated by the summation of the maximum weight value resulting from all possible immediate successor nodes, the weight of an edge, and the rank value of that successor node (i.e., the node has the maximum weight value), as represented in equation [2]. The tasks are arranged in descending order in the list according to their rank value. Then, the tasks are scheduled on the processors of the underline heterogeneous machine to guarantee earliest finish time [3]. Tie is resolved randomly [9].

$$\mathrm{avg}[t] = \left(\sum_{i=m}^{i=0} c_i\right) \setminus m \tag{1}$$

where c_i is the execution time of task on each processor

$$r_u(i) = f_1\left(w_i^0, \ldots, w_i^m, \ldots, w_i^{m-1}\right) + \max_{j \in Si}\left(f_2\left(C_{ij}^{00}, \ldots, C_{ij}^{mm\prime}, \ldots, C_{ij}^{M-1,M-1}\right) + r_u(j)\right) \tag{2}$$

where W_i^m is the computation cost of task i on machine m, $0 < m < M$, S_i is the set of the immediate successor of task i, and $C_{ij}^{mm\prime}$ is the communication cost between nodes i and j when i executed by machine m and j by machine m', $0 \leqslant m$, $m' < M$ [3]. It is assumed that when i and j are executed by the same machine the communication cost is zero. The function f_1 returns a value which is dependent on the computation cost of a given task on every machine, and f_2 returns a value which is dependent on the communication cost between two given tasks may execute [5].

The pseudo code of the ranking method of HEFT algorithm is as following:

Ranking algorithm 1.1: Calculate rank value to give priority to sort tasks to be executed

```
1.  for each task in DAG calculate average execution time on all
    processors
2.    if task is last task
3.    Rank of task = average of the task
4.    else
5.       Rank of task = average of
    task + max(rank(predecessors)) + channel weight
6.    end if
7.  end for
```

The pseudo code for assigning tasks to available processors of the underline machine according to HEFT algorithm is as follows:

Mapping method 1.2: assign tasks to available machines

```
1.  for each task in list of ordered tasks
2.      if task is first task in the list then
3.          map task to processor with minimum execution time
4.      else
5.      if task and its predecessor on the same processor p_j
6.      comm_time = 0
7.      else
8.      comm._time = communication time between two nodes
9.      end if
10.   for each processor
11.  Task_execution_time = excution_time of task on
     processor + comm_time + predecessor_excution time
12.   end for
13.  end for
14.  map task to processor with minimum total_execution_time
```

According to the work in this paper, the HEFT algorithm has been modified to improve the load balancing according to following equation:

1. determine the limit number of tasks can be mapped for each processor by

$$L = \text{number of tasks/number of processors.} \quad (3)$$

2. Assigning the tasks to each processors according to the following equation

$$\text{Number of tasks on each processor} \leqslant L. \quad (4)$$

Unfortunately, this modification increases the total finish time which minimize the performance of HEFT algorithm. This drawback has been overcome by introducing the duplication based scheduling [4,9].

3. The Triplet algorithm

The main concepts of the Triplet algorithm is that the application subtasks, and the processors in the underline system are grouped into clusters and mapping the subtasks clusters onto processors clusters to minimize communication overhead [5].

The application subtasks are grouped based on interconnection to minimize communication overhead. On the other hand, the processors of underline system which have the same characteristics (network, processor speed, etc.) are grouped into clusters (i.e. convert the heterogeneous underling system into set of homogenous subsystem).

The Triplet algorithm phases are:

Task clustering phase

Tasks are grouped into clusters in order to overcome communication overhead. The pseudo code of task cluster is as follows:

Method 2.1: Tasks clustering [5]

```
1.  Put each task in a cluster
2.  Generate the list of all the triplets
3.  Sort the triplets by decreasing degree and by decreasing
      Amount of communication
4.  for each triplet do
5.    if geometric or temporal criterion is fulfilled then
6.    merge the two clusters
7.    end if
8.  end for
```

According to method 2.1: each task in a cluster has id. The clusters tasks which are belong to a path of length **2** in the task graph are grouped, where a path of length 2 is composed of three tasks and is called a **triplet** [5]. Once all triplets in the task graph are generated, triplets are sorted in order to consider large communicating edges first. Triplets are sorted first by their degree and second by their decreasing amount of communication produced by its three tasks. A geometric approach can be used to find any possibilities to minimize number of clusters.

Processors clustering phase

In this phase, the processors which have the same characteristics are grouped into cluster in order to make each cluster as a set of homogeneous system. On other hand, if the powers of processors are completely different, then each processor can be considered as a cluster.

Mapping phase

In this phase, the tasks clusters are mapped to the processor clusters. The pseudo code of the mapping function is as follows [5]:

Method 2.2: Mapping task's clusters to clusters of machines

```
1.  Sort machine's clusters by decreasing CPU power
2.  Determine a maximum load for each cluster of machine
3.  Sort task's clusters by degree and amount of external
    communications
4.    for each task's cluster in the order do
5.      if the number of tasks assigned to current
        Cluster of machines exceed its load. Then.
        switch to next machine cluster.
6.    end if
7.  Assign current task's cluster to the machine having the best
    completion time.
8.  end for
```

Initially, the processors clusters are sorted according to its CPU power, and task clusters are sorted according to its degree and the amount of external communication. Then, the tasks are mapped to the processors with considering the maximum number of tasks on each processor to satisfy the load balancing [7,10].

4. The developed Cluster Based HEFT with Duplication (CBHD) algorithm

Generally, the HEFT algorithm is conceded simple list scheduling algorithm, which achieves high performance and very good makespane, but its drawback is that there is no load balancing among processors of the underline system. On other hand, the Triplet Cluster algorithm tries to determine the number of tasks can be loaded by each processor, this can help in avoiding overload on some processors but it is not prevent unfair in distributing tasks between processors due to the sleek time [3,5,11].

According to the developed CBHD algorithm, these above drawbacks have been overcame by amalgamating the HEFT, and Triplet algorithms by clustering tasks according to Triplet algorithm with merging the produced clusters to minimize communication over heads, and then, sorting tasks in each cluster to be executed according to rank function of the HEFT algorithm.

Extra modification has been added by introducing the duplication phenomena [4,9]. According to the duplication phenomena, tasks are executed on more than one processor based on the following conditions:

1. The duplicated task can either be the first or the last task in the task cluster.
2. The first task in task clusters can be duplicated only if:
   The task execution time on processor $\mathbf{p_j}$ < the execution time on processor $\mathbf{p_i}$ + minimum communication time between task and its successors.
3. The last task in task clusters can be duplicated only if:
   The task finish execution time on processor $\mathbf{p_i} \leqslant$ task finish execution time on $\mathbf{p_j}$.

where processor $\mathbf{p_i}$ is the main processor for executing the task during the mapping stage, and $\mathbf{p_j}$ is the processor which runs the duplicated task. Once the duplicated task is finished on any processor, the tasks which depend on it will start executing immediately, and all the machines will be working in the same time. So the developed algorithm achieves full utilization and load balancing among processors in the underline system, and keeps minimum makes pane.

The pseudo code of Cluster Based HEFT with Duplication (CBHD) algorithm is represented as follows:

```
1. for each node calculate rank value applying Rank method 1.1
   to prioritize the tasks for execution
2. end for
3. apply clustering method 2.1 in order to grouping tasks in
   clusters
4. merge the cluster to minimize communication over heads
5. for each cluster of tasks
6.    order tasks according to rank value calculated in step 1
7.       map cluster of tasks to machines gives minimum total
   execution time
8.    if machine exceed the limit determined for the number of
   tasks load
9.    map task to the next machine with minimum execution time
10.   End if
11. If task is first item in task cluster
       If the task i execution time on processor pj < task execution
   time on processor pi + minimum communication time between
   task and its successors.
12 Duplicate task on processor with minimum execution time
13    End if
14 End if
15    If task is last item in cluster task
16.      If the task finish execution time on processor pi ⩽ task
   finish execution time on pj
17.      Duplicate task on processor with minimum execution time
18.      End if
19.   End if
20. End for
```

To evaluate the performance of the developed algorithm, a comparative study has been done among it, the Triplet, and the HEFT algorithms by using the following measures:

1. Makespane (maximum needed time to complete the execution of all tasks)

$$\text{Makespane} = \max_{t_i \in T}(c_i) \tag{5}$$

2. $$\text{Utilization} = \frac{\sum \text{Machine Busy Times}}{\text{Overall System Time}} * 100 \tag{6}$$

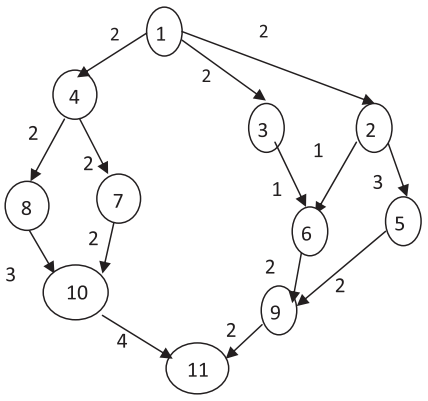

Figure 1 An application DAG of example 1.

3. Load balancing (limited number of tasks can be executed on a processor)

$$L = \text{number of tasks/number of processors} \tag{7}$$

5. Performance evaluation

To evaluate the developed CBHD algorithm, a comparative study has been done among it, the HEFT, and the Triplet algorithms by considering the following examples.

A simulator called Distributed Algorithm Simulator[1] has been used. This simulator involves building a graphical tool that enables to simulate different network topology like linear, ring and mesh, or any other network topology supported with message passing architecture. Additional experiment outcome analysis features could also be supported in this simulator by the programmer. This simulator is executed using computer Pentium 4, CPU 1.86 GHz, and 504 MB of RAM.

5.1. Example 1

By considering the following DAG of an application with communication cost (see Fig. 1). The computation cost for each node on four processors in the underline heterogeneous system is represented in Table 1.

5.1.1. The HEFT algorithm implementation

First: The rank value for each node in the DAG is calculated using Eqs. (2) and (1), (see Table 2).

Second: The tasks are ordered in descending order for execution according to its rank. The resulting list $\{1,2,4,3,7,8,6,5,10,9,11\}$.

Third: The nodes are mapped according to its rank to the processors to obtain the minimum makespane (see Fig. 2).

According to results in Fig. 2, the makespane = 27, there is no load balancing between the processors where P1 is over loaded with respect to other processors. The processor utilization is calculated using Eq. (5). The implementation results of the HEFT algorithm are summarized in Table 3.

[1] Developed by Yussuf Abu Shaaban and Jane Hillston, School of Informatics, Edinburgh, University, UK, 2003.

Table 1 Computation cost for each node on four processors.

Task	P1	P2	P3	P4
1	4	4	4	4
2	5	5	5	5
3	4	6	4	7
4	3	3	3	3
5	3	5	3	4
6	3	7	2	2
7	5	8	5	5
8	2	4	5	3
9	5	6	7	5
10	3	7	5	2
11	5	6	7	8

Table 2 The upward ranking for workload in Fig. 1.

Task	P1	P2	P3	P4	Average weight	Rank
1	4	4	4	4	4	34
2	5	5	5	5	5	28
3	4	6	4	7	5.25	26.50
4	3	3	3	3	3	27.50
5	3	5	3	4	3.75	20
6	3	7	2	2	3.5	20.25
7	5	8	5	5	5,75	22.50
8	2	4	5	3	3.5	21.25
9	5	6	7	5	5.75	14.25
10	3	7	5	2	4.25	14.75
11	5	6	7	8	6.5	6.5

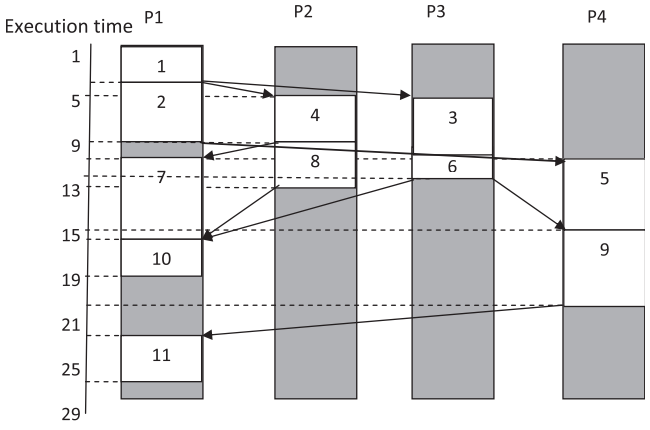

Figure 2 Mapping nodes to processors.

According to the results in Table 3, it is found that there is no load balancing between processors, and the utilization of the processors, p2, p3, and p4 is poor.

5.1.2. The Triplet algorithm implementation

By applying the Triplet algorithm using the DAG of example 1 in Fig. 1.

First: Tasks are grouped into clusters as illustrated previously in Section 3 by applying **method 2.1** (see Fig. 3).

Second: According to algorithm 2.2., mapping clusters of tasks to processors of the underline system (see Fig. 4).

The implementation results of the Triplet algorithm are summarized in Table 4.

According to the results in Table 4, it is found that the HEFT algorithm's makespane better than the Triplet algorithm's makespane by 6%, but the Triplet algorithm is better than the HEFT algorithm by 50% for load balancing.

5.1.3. The developed Cluster Based HEFT with Duplication (CBHD) algorithm

The developed CBHD algorithm has been implemented into two versions; without duplication, and with duplication using DAG of the example 1.

5.1.3.1. CBHD algorithm without duplication. The nodes of the DAG are grouped into clusters by applying **method 2.1**. Then nodes in each cluster are sorted descending according to the nodes ranks which are calculated based on the HEFT algorithm (see Table 2). The produced clusters are presented in Fig. 5.

After that, the clusters of tasks are mapped onto processors of the underline system (see Fig. 6).

The implementation results of the developed (CBHD) algorithm without duplication are summarized in Table 5.

According to the results in Table 5, it is found that the developed CBHD algorithm without duplication satisfies the load balancing than the HEFT algorithm by 50%. On other hand the makespane is not the optimum where it is grater than the makespane of the HEFT algorithm implementation. So we have to apply the duplication to improve the performance of CBHD algorithm.

5.1.3.2. CBHD algorithm with duplication. The executions of some tasks have been duplicated in different processors such that the dependence tasks can be started as soon as one of the duplicated tasks is finished (see Fig. 7).

As shown in Fig. 7, task 1 is duplicated according to the duplication rules, let task 2 starts execution early but without effect in the total execution time. On the other hand, duplication of task 11 minimizes the total execution time to 26 instead of 29. The implementation results of the developed CBHD algorithm with duplication are summarized in Table 6.

According to the implementation results of the developed CBHD with duplication algorithm, it is found that the developed algorithm improves performance with respect to the

Table 3 Performance evaluation of HEFT algorithm.

	Makespane = 27			
	P1	P2	P3	P4
Processors utilization	81%	25%	25%	25%
Load balancing	Overload	No load	No load	No load
Sleek time (time the machine is idle) by ms	4 (most of time working)	20 (most of time idle)	21 (most of time idle)	18 (most of time idle)

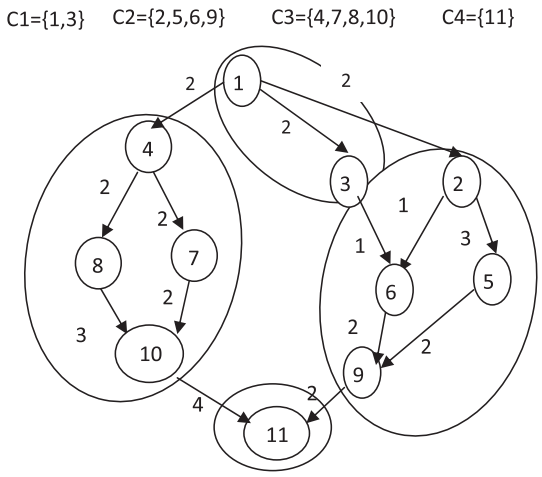

Figure 3 Clusters of grouped tasks.

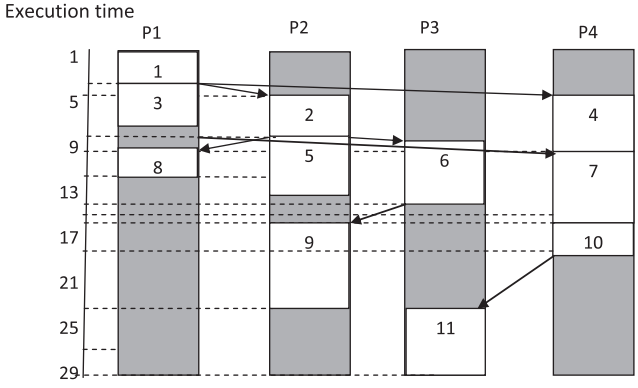

Figure 4 Mapping cluster of tasks to processors using Triplet algorithm.

HEFT algorithm by 50% for the load balancing, 9% for the processors utilization, and in the same time, it achieves minimum makespane = 26.

The performance evaluation results of example 1 using the HEFT, the Triple, and the developed CBHD with and without duplication are listed in Figs. 8–10 respectively.

5.2. Example 2

By considering the following DAG of an application with communication cost (see Fig. 11). The computation cost for each node on three processors in the underline heterogeneous system is represented in Table 7.

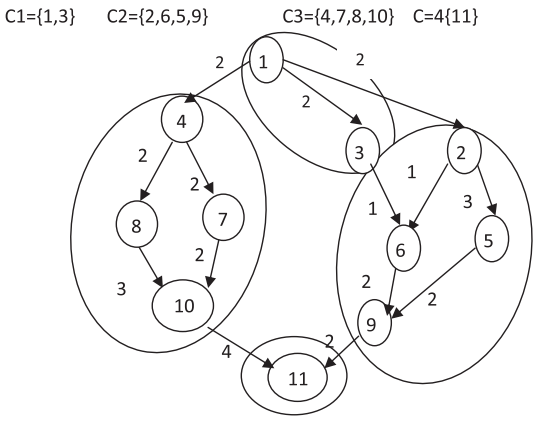

Figure 5 Clusters of grouped tasks.

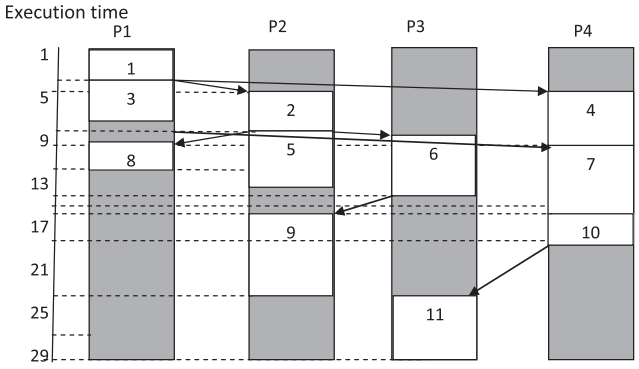

Figure 6 Mapping cluster of tasks to processors using CBHD without duplication.

5.2.1. The HEFT algorithm implementation

First; the rank value for each node in the DAG of example 2 is calculated using Eqs. (1) and (2), (see Table 8).

Second: The tasks are ordered descending for execution according to its rank the resulting list $\{1,4,3,2,6,5,8,9,7,10\}$.

Third: The nodes are mapped according to its rank to the processors to obtain the minimum makespane (see Fig. 12)

According to results (see Fig. 12), the makespane = 169, there is no load balancing between the processors where P3 is over loaded with respect to other processors.

The implementation results of the HEFT algorithm are summarized in Table 9.

According to the results in Table 9 it is found that the makespane = 169 there is no load balancing between processors, and the utilization of the processors p1 and p2 is poor.

Table 4 Performance of Triplet algorithm.

	Makespane = 29			
	P1	P2	P3	P4
Processors utilization	34%	55%	31%	34%
Load balancing	Good load	Good load	No load	Good load
Sleek time by ms	19 ms (sometimes idle)	13 ms (sometimes idle)	20 ms (sometimes idle)	19 ms (sometimes idle)

Table 5 Performance of developed CBHD algorithm without duplication.

	Makespane = 29			
	P1	P2	P3	P4
Processor utilization	34%	55%	31%	34%
Load balancing	Good load	Good load	No load	Good load
Sleek time (time the machine is idle) by ms	19 ms (sometimes idle)	13 ms (sometimes idle)	20 ms (sometimes idle)	19 ms (sometimes idle)

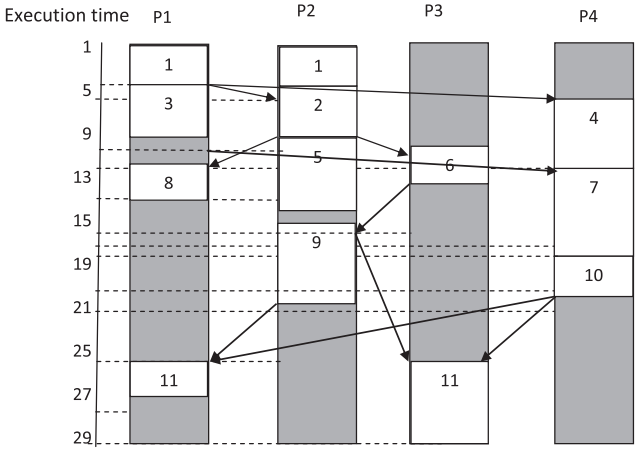

Figure 7 Mapping cluster of tasks ordered according to its rank to processors with duplication.

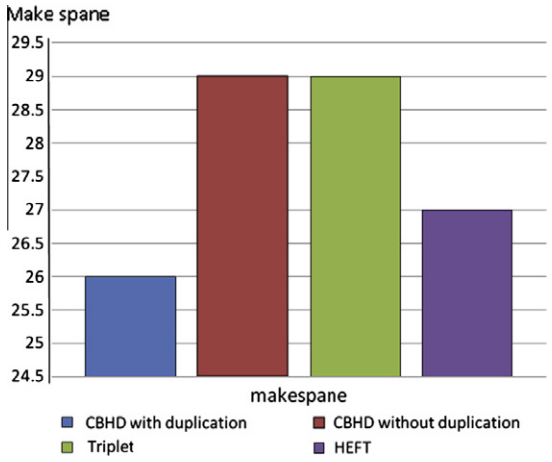

Figure 8 Comparison of makespane of three algorithms.

5.2.2. The Triplet algorithm implementation

First: Tasks are grouped into clusters according to clustering **method 2.1** (see Fig. 13).

Second: The clusters of tasks are mapped to processors of the underline system (see Fig. 14).

The implementation results of the Triplet Cluster algorithm are summarized in Table 10.

According to the results in Table 10,it is found that The Triplet algorithm has nearly makespane as the HEFT algorithm, but the Triplet algorithm satisfies better load balancing between processors and utilization than the HEFT algorithm.

5.2.3. The developed Cluster Based HEFT with Duplication (CBHD) algorithm

5.2.3.1. CBHD algorithm without duplication. The nodes of the DAG are grouped into clusters by applying **method 2.1**, then nodes in each cluster are sorted descending according to the nodes ranks which are calculated based on the HEFT algorithm (see Table 8). The produced clusters are presented in Fig. 15.

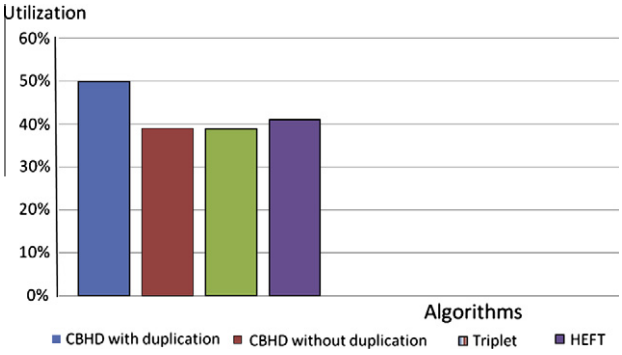

Figure 9 Comparison of utilization of three algorithms.

Table 6 Clustering Based HEFT with Duplication algorithm (CBHD).

	Makespane = 26			
	P1	P2	P3	P4
Processors utilization	57%	70%	34%	38%
Load balancing	Loaded	Loaded	Accept load	Good load
Sleek time (time the machine is idle) by ms	14 ms (some times idle)	9 ms (most of time working)	20 (most of time idle)	18 (sometimes idle)

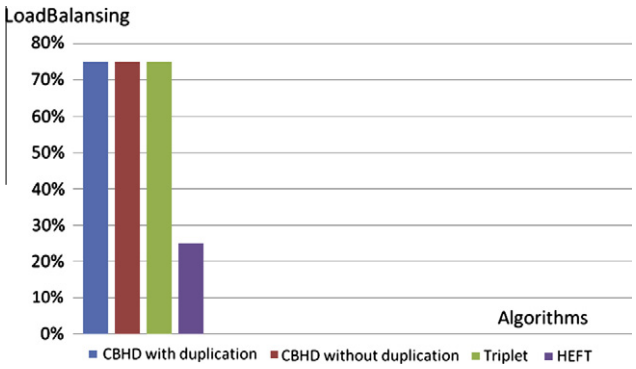

Figure 10 Comparison of load balancing of three algorithms.

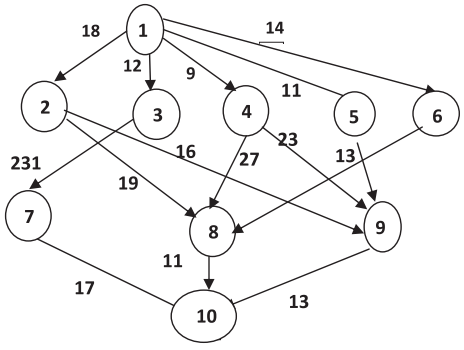

Figure 11 An application DAG of example2.

Table 7 Computation cost for each node on three processors.

Task	P1	P2	P3
1	37	39	27
2	30	20	24
3	21	21	28
4	35	38	31
5	27	24	30
6	29	37	20
7	22	24	30
8	37	26	37
9	35	31	26
10	33	37	21

Table 8 The upward ranking for workload in Fig. 11.

Task	P1	P2	P3	Average	Rank
1	37	39	27	34.3	165.3
2	30	20	24	24.6	117.6
3	21	21	28	23.3	118.3
4	35	38	31	34.7	135.7
5	27	24	30	27	112
6	29	37	20	106	117
7	22	24	30	25.3	72.3
8	37	26	37	33.3	74.3
9	35	31	26	30.6	73.6
10	33	37	21	30.3	30

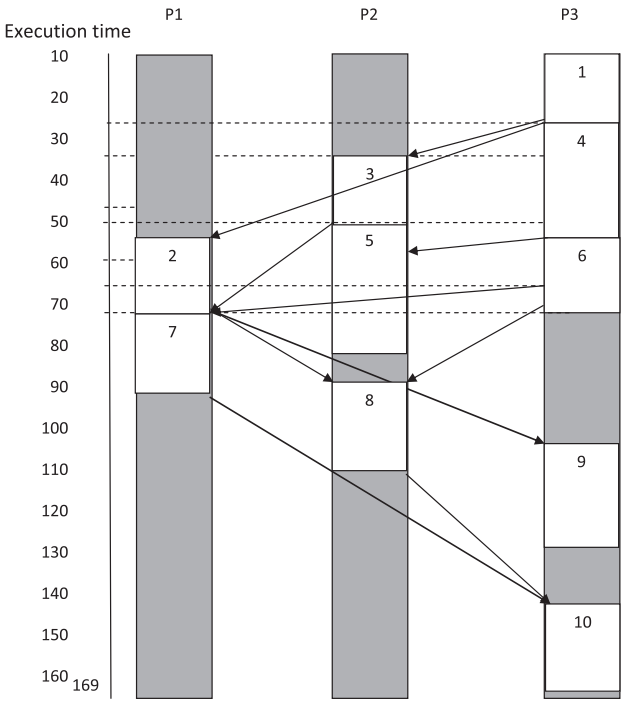

Figure 12 Mapping nodes according to its rank to three processors of underline system.

Table 9 Performance evaluation of HEFT Algorithm.

	Makespane = 169		
	P1	P2	P3
Processors utilization	30%	43%	74%
Load balancing	No load	Acceptable load	Over loaded
Sleek time (time the machine is idle) by ms	117 ms (most of time idle)	98 ms (sometimes idle)	44 ms (most of time is working)

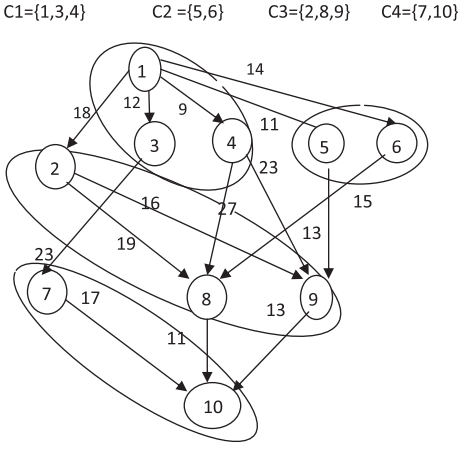

Figure 13 Grouped tasks into clusters.

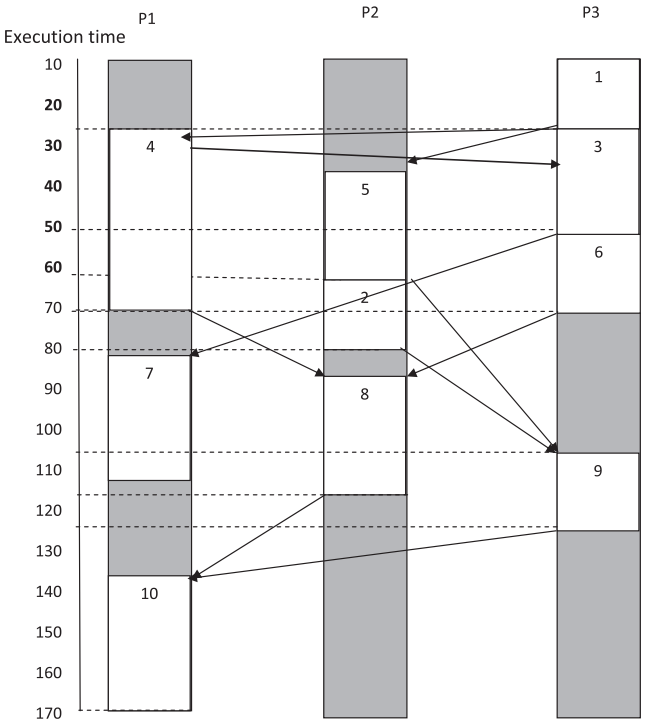

Figure 14 Mapping cluster of tasks to three processors of underline system using triplet.

The clusters of tasks are mapped onto processors of the underline system (see Fig. 16).

The implementation results of the developed (CBHD) algorithm without duplication are summarized in Table 11.

According to the results in Table 11, it is found that the developed CBHD without duplication algorithm satisfies load balancing and processors utilization than the HEFT algorithm. On the other hand the makespane is not the optimum where it is grater than the makespane of the HEFT algorithm implementation. So we have to apply the duplication to improve the performance of CBHD algorithm.

5.2.3.2. CBHD algorithm with duplication. The executions of some tasks have been duplicated in different processors such that the dependence tasks can be started as soon as one of the duplicated tasks is finished (see Fig. 17).

As shown in Fig. 17, task 9 is duplicated such that task 10 can execute in the same processor which minimizes the communication time with keeping the load balancing between processors.

The implementation results of the developed CBHD algorithm with duplication are summarized in Table 12.

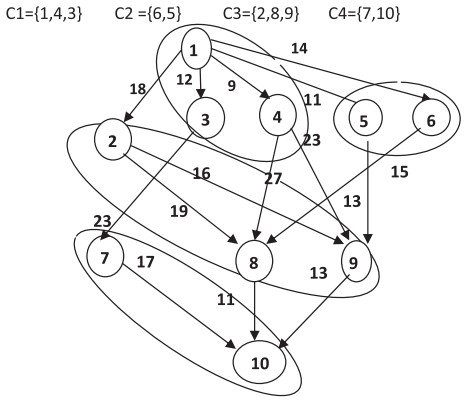

Figure 15 Grouped ranking tasks into clusters.

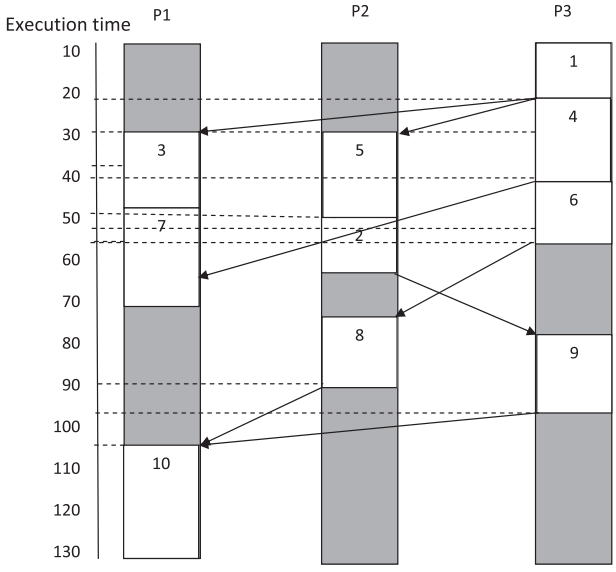

Figure 16 Mapping ordered nodes according to its rank to three processors of underline system.

According to the implementation results of the developed CBHD with duplication algorithm, the makespane = 166 which means that the performance is improved with respect to the HEFT algorithm by 2% for makespane, 33% for load balancing, and 8% for processors utilization. Comparing to Triplet algorithm the performance of the developed CBHD algorithm with duplication is improved by 2.35% for

Table 10 Performance of Triplet algorithm.

	Makespane = 170		
	P1	P2	P3
Processors utilization	52%	41%	59%
Load balancing	Good load	Good load	Good load
Sleek time (time the machine is idle) by ms	80 ms (sometimes idle)	100 ms (sometimes idle)	69 ms (most of time is working)

Table 11 Performance of CBHD Algorithm without duplication.

	Makespane = 170		
	P1	P2	P3
Processors utilization	45%	42%	61%
Load balancing	Acceptable load	Acceptable load	Acceptable load
Sleek time (time the machine is idle) by ms	94 ms (most of time idle)	95 ms (most of time is idle)	66 ms (sometimes idle)

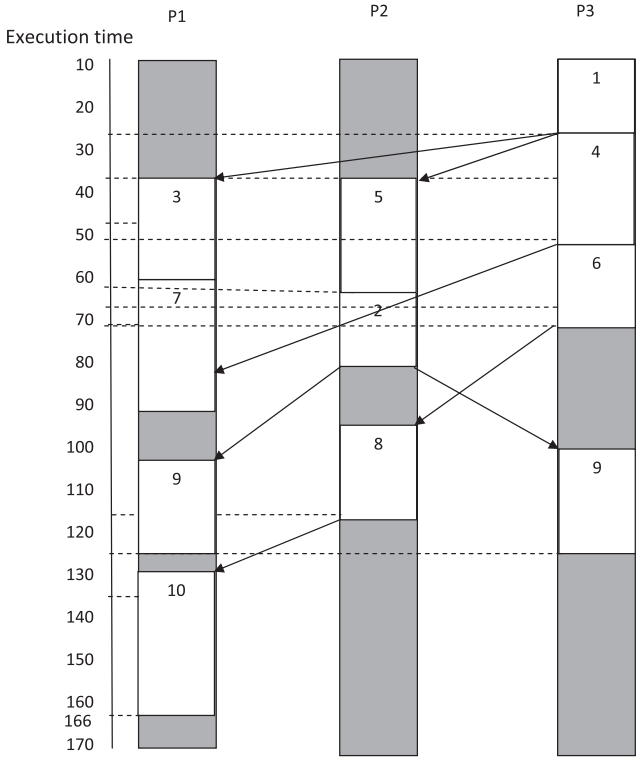

Figure 17 Mapping ordered nodes according to its rank to three processors of underline system with duplication.

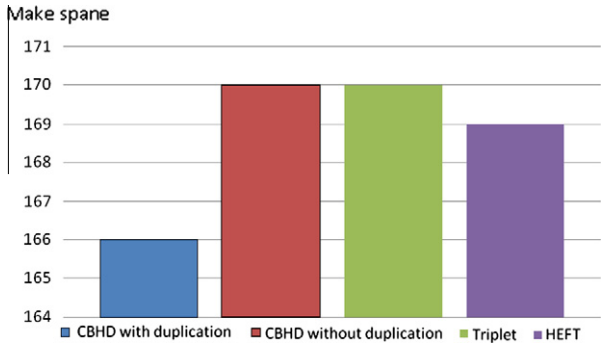

Figure 18 Comparison of makespane of three algorithms.

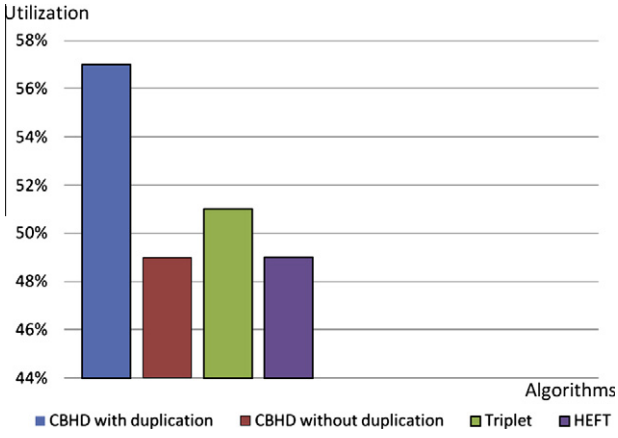

Figure 19 Comparison of utilization of three algorithms.

makespane, and 6% for processors utilization keeping with 100% load balancing.

The results of performance evaluation of example 2 are summarized in Figs. 18–20 respectively.

6. Conclusions

According to the work in this paper, the Cluster Based HEFT with and without Duplication (CBHD) algorithm is proposed. The developed CBHD algorithm is considered an amalgamation of the Triplet and the HEFT algorithms, where the tasks are divided into clusters according to the triplet algorithm, and the tasks in each cluster are ordered according to its rank based on the HEFT algorithm. To evaluate the performance of the developed CBHD algorithm, a comparative study for the HEFT, Triplet Cluster, and the developed CBHD algorithms has been done. According to the performance evaluation, it is found that the HEFT algorithm is considered the

Table 12 Clustering Based HEFT with Duplication algorithm (CBHD).

	Makespane = 166		
	P1	P2	P3
Processors utilization	67%	42%	62%
Load balancing	Acceptable load	Acceptable load	Acceptable load
Sleek time (time the machine is idle) by ms	55 ms (most of time working)	96 ms (most of time is idle)	62 ms (sometimes idle)

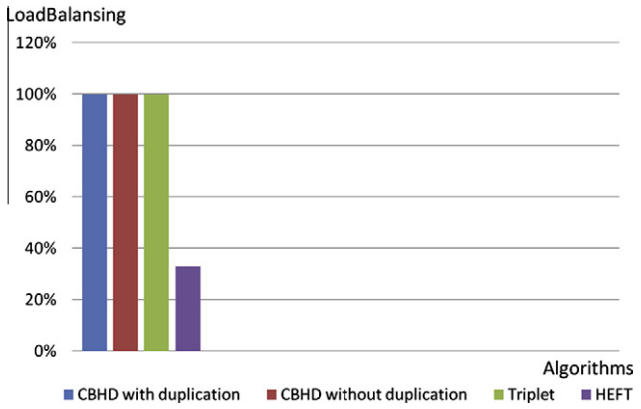

Figure 20 Comparison of load balancing of three algorithms.

simple dynamic task scheduling with less computation complexity than that the Triplet algorithm. On the other hand, the produced makespane of the Triplet algorithm in most cases is the same or larger than the HEFT algorithm, but the Triplet Cluster algorithm almost satisfies both the load balancing between processors, and processors utilization. According to the comparative study among the developed CBHD algorithm, the Triplet algorithm and the HEFT algorithm, it is found that the developed CBHD algorithm outperforms the HEFT and Triplet algorithm by decreasing the makespane by 2.5%. It also achieves better load balancing than the HEFT algorithm by 70%, and it increases processors utilization by 10% with respect to the HEFT and Triplet algorithms. The CBHD algorithm has developed into two versions, without and with duplication. The CBHD algorithm without duplication may increase the makespane which results in lower performance. The CBHD Algorithm with duplication has achieved minimum makespane, maximum utilization and load balancing which is considered the main performance parameter in dynamic environment. Generally, the developed CBHD is almost satisfied minimum makespane, load balancing and processors utilization.

References

[1] Ucar Bora, Aykanat Cevdet, Kaya Kamer, Ikinci Murat. Task assignment in heterogeneous computing systems. J Parallel Distrib Comput 2005;66:32–46.
[2] Radulescu A, van Gemund AJC. Fast and effective task scheduling in heterogeneous systems. In: 9th Heterogeneous computing, Workshop; 2000. p. 299–238.
[3] Zhao Rizos Henan, Sakellariou Rizos. An investigation into rank function of the heterogeneous earliest finish time (HEFT) algorithm. University of Manchester, UK: Department of Computer Science; 2003.
[4] Bajaj R, Agrawal PD. Improving scheduling of tasks in a heterogeneous environment. IEEE Trans Parallel Distrib Syst 2004;15(2):107–18.
[5] Cirou Bertrand, Jeannot Emmanuel. Triplet: a clustering scheduling algorithm for heterogeneous systems. In: IEEE symposium on reliable distributed systems; 2001. p. 231–6.
[6] Omara Fatma A, Arafa Mona M. Genetic algorithms for task scheduling problem. Parallel Distrib Comput J 2010;70(1):13–22.
[7] Rudolph Larry, Slivkin Miriam, Upfal Eli. A simple load balancing scheme for task allocation in parallel machines. In: Proceedings of the third annual ACM symposium on parallel algorithms and architectures; 1991. p. 237–45.
[8] Nour Mohamed, Omara Fatma A. Performance evaluation of parallel task scheduling algorithms for heterogeneous processors. Egypt Inform J 2005;6(1):98–122.
[9] Ahmed Ishfaq. On exploiting task duplication in parallel program scheduling. IEEE Trans Parallel Distrib Syst 1998;9:872–96.
[10] Boivin Simon, Gendron Bernard, Pesant Gilles. A load balancing procedure for parallel constraint programming, CIRRELT-2008-32; 2008.
[11] Iqbal Saeed, Gupta Rinku, Fang Yung-Chin. Job scheduling in HPC clusters. Dell Power Solution; 2005, p. 133–6.