Performance Enhancement of Scheduling Algorithm in Heterogeneous Distributed Computing Systems

Aida A. NASR
Computer Science & Eng. Dept.,
Faculty of Electronic Engineering,
Menoufia Uni., Menouf 32952,
Egypt

Nirmeen A. EL-BAHNASAWY
Computer Science & Eng. Dept.,
Faculty of Electronic Engineering,
Menoufia Uni., Menouf 32952,
Egypt

Ayman EL-SAYED
Computer Science & Eng. Dept.,
Faculty of Electronic Engineering,
Menoufia Uni., Menouf 32952,
Egypt

***Abstract*—Efficient task scheduling is essential for obtaining high performance in heterogeneous distributed computing systems. Some algorithms have been proposed for both homogeneous and heterogeneous distributed computing systems. In this paper, a new static scheduling algorithm is proposed called Node Duplication in Critical Path (NDCP) algorithm to schedule the tasks efficiently on the heterogeneous distributed computing systems. The NDCP algorithm focuses on reducing the makespan and provides better performance than the other algorithms in metrics of speedup and efficiency. It consists of two phases, priority phase and processor selection phase. From the theoretical analysis of the NDCP algorithm with other algorithms for a Directed Acyclic Graph (DAG), the better performance is observed.**

Keywords—static task scheduling; heterogeneous distributed computing systems; Meta-heuristic algorithms

I. INTRODUCTION

The availability of high-speed networks and diverse sets of resources lead to a platform, called as heterogeneous platform. Such a platform contains interconnected resources with different computing capabilities and different computing speeds. To run an application in this heterogeneous environment, several issues need to be considered such as partitioning the application, scheduling the tasks; etc. It is referred to such a system as Heterogeneous Distributed Computing System (HDCS). In recent years, HDCS has emerged as a popular platform to execute computationally intensive applications with diverse computing needs [1].

Task scheduling is of vital importance in HDCS since a poor task-scheduling algorithm can undo any potential gains from the parallelism presented in the application. In general, the objective of task scheduling is to minimize the completion time of a parallel application by properly mapping the tasks to the processors. There are typically two categories of scheduling models: static and dynamic scheduling. In the static scheduling case, all information regarding the application and computing resources such as execution time, communication cost, data dependency, and synchronization requirement is assumed available a priori. Scheduling is performed before the actual execution of the application [2, 3]. On the other hand, in the dynamic mapping a more realistic assumption is used. Very little a priori knowledge is available about the application and computing resources. Scheduling is done at run-time [4]. In this paper, it is focused on static scheduling. Static scheduling has three categories: *list-based, clustering* and *duplication based*.

List-scheduling algorithms contain two phases: *a task prioritization phase*, and a *machine assignment phase*. In *prioritization phase,* the algorithms assign a certain priority that is computed, to node in the DAG. In *machine assignment phase*, each task depending on its priority is assigned to machine that minimizes the cost function [5-9]. Examples of list-based algorithms are Heterogeneous Earliest Finish Time (HEFT) and Critical Path on Processor (CPOP) [10]. Another static scheduling category is task duplication based algorithms, in which tasks are duplicated on more than one processor to reduce the waiting time of the dependent tasks. The main idea behind duplication based scheduling is to utilize processor idling time to duplicate predecessor tasks. This may avoid transfer of results from a predecessor, through a communication channel, and may eliminate waiting slots on other processors and reduce the communication overheads [11,12]. An example for duplication algorithms is Heterogeneous Critical Node First (HCNF) and Scalable Task Duplication Based Scheduling (STDS) [13,14].

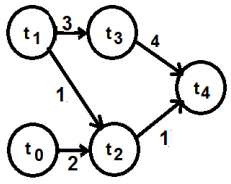
In this paper, a new algorithm called Node Duplication in Critical Path (NDCP) is developed for static task scheduling for the HDCS with limited number of processors. The motivation behind this algorithm is to generate the high quality task schedule that is necessary to achieve high performance in HDCS. The developed algorithm is based on critical path method to give each node a priority, and the duplication algorithm to minimize communication overheads. Finally, idle time is decreased in proposed algorithm.

The remainder of this paper is organized as follows. Section II discusses problem definition. Section III gives an overview of the related work. Section IV presents our developed *NDCP* algorithm with examples. Section V discusses the results and in section VI, conclusions are given.

II. PROBLEM DEFINITION

Task scheduling for HDCS is the problem of assigning the tasks of a parallel application to the processors of a HDCS, which have diverse capabilities, and specifying the start execution time of each task. This must be done in a way that respects the precedence constraints among tasks. An efficient schedule is one that minimizes the total execution time, or the schedule length, of the parallel application [15-23].

The models of HDCS [24] and the model of application to be considered in this work can be described as follows. By using DAG, the parallel application is represented. DAG is defined by the tuple (T,E), where T is a set of n tasks and E is a set of e edges. Each $t_i \in T$ represents a task in the parallel application, which in turn is a set of instructions that must be executed sequentially in the same processor without interruption. Each edge $(t_i, t_j) \in E$ represents a precedence constraint, such that the execution of $t_j \in T$ cannot be started before $t_i \in T$ finishes its execution. If $(t_i, t_j) \in E$, then t_i is a parent of t_i and t_j is a child of t_i. A task with no parents is called an entry task t_{entry}, and a task with no children is called an exit task t_{exit}. Each edge $(t_i, tj) \in E$ has a value that represents the estimated inter-task communication cost required to pass data from the parent task t_i to the child task t_j. Because tasks might need data from their parent tasks, a task can start execution on a processor only when all data required from its parents become available to that processor; at that time the task is marked as ready. The speed of the inter-processor communication network is assumed to be much lower than the speed of the intra-processor bus. Therefore, when two tasks are scheduled on the same processor the communication cost between these tasks can be ignored. The HDCS is represented by a set P of m processors that have diverse capabilities. The n×m computation cost matrix C stores the execution costs of tasks. Each element $c_{i,j} \in C$ represents the estimated execution time of task t_i on processor p_j. Precise calculation of the running times of the tasks on the processors is unfeasible before running the application. All processors in the HDCS are assumed to be fully connected. Communications between processors occur via independent communication units; this allows for concurrent execution of computation of tasks and communications between processors. After scheduling all the tasks of a parallel application on the processors of a HDCS, the schedule length is defined as the longest finish time of the HDCS processors. Fig. 1 presents an example of a parallel application consisting of five tasks and a HDCS with two processors, where the application is represented as a DAG and the execution costs estimated for the five tasks on the HDCS are shown as a computation cost matrix.



(a) DAG

Task	P_0	P_1
t_0	7	8
t_1	6	9
t_2	5	8
t_3	2	3
t_4	2	4

(b) Computation Cost Matrix

Fig. 1. Example of a DAG and Computation Cost Matrix

Definition (1) Critical Path *(CP)*: *CP* of a *DAG* is the longest path from the entry task to the exit task in the graph.

Definition (2) $EST(t_i, P_j)$ [10]: Denotes the Earliest Start Time of a task t_i on a processor P_j and is defined as shown in Equation 1.

$$EST(t_i, P_j) = max\{ T_{Available}(P_j), max\{AFT(t_k) + c_{k,i}\} \} \text{--(1)}$$

Where $T_{Available}(P_j)$ is the earliest time at which processor P_j is ready. $AFT(t_k)$ is the Actual Finish Time of a task t_k (where t_k is the parent of task t_i and $k=1, 2, \ldots, n$) on the processor P_j. $c_{k,i}$ is the communication cost from task t_k to task t_i, $c_{k,i}$ equal zero if the predecessor task t_k is assigned to processor P_j. For the entry task, $EST(t_{entry}, P_j) = 0$.

Definition (3) $EFT(t_i, P_j)$ [10]: Denotes the Earliest Finish Time of a task t_i on a processor P_j and is defined as shown in Equation 2.

$$EFT(t_i, P_j) = EST(t_i, P_j) + w_{i,j} \text{---------- (2)}$$

Which is the Earliest Start Time of a task t_i on a processor P_j plus the computational cost $w_{i,j}$ of t_i on a processor P_j.

Definition (4) Data Ready Time (DRT): is the idle time waited by a t_i on processor p_j.

Definition (5) Maximum Parent (MP): maximum parent of task t_i is a parent task t_k such that the value of $EFT(t_k, p_m) + c(t_k, t_i)$ is the largest among all t_i's parent tasks.

Definition (6) Very Important Task (VIT): is the task that belongs to the critical path of DAG.

III. Related Work

In this section, it is given an overview of some algorithms, specifically list-based scheduling algorithms.

A. Heterogeneous Earliest Finish Time

The *HEFT* algorithm executes in two phases: a task-prioritizing phase and processor selection phase [10]. In task prioritizing phase, the algorithm selects the task with the highest upward rank at each step. Upward rank is given by Equation 3.

$$Ranku = \overline{w_i} + max_{n_j \in succ(n_i)}(\overline{c_{i,j}} + rank_u(n_j)) \text{--------(3)}$$

Where $succ(n_i)$ is the set of immediate successors of task n_i, $\overline{c_{i,j}}$ is the average communication cost of *edge(i,j)*, and $\overline{w_i}$ is the average computation cost of task n_i .In processor selection phase, the selected task is assigned to the processor which minimizes its earliest finish time with an insertion-based approach. The algorithm has an $O(n^2p)$ time complexity for n nodes and p processors.

B. Critical Path On Processor Algorithm

The *CPOP* algorithm consists of two phases: prioritizing phase and processor selection phase [10]. In task prioritizing phase, the algorithm selects the task with the highest of upward rank + downward rank value at each step. Downward rank can be calculated by Equation 4.

$$Rankd(ni) = max_{n_j \in pred(n_i)}(\overline{c_{j,i}} + \overline{w_j} + rank_d(n_j)) \text{--(4)}$$

Where $pred(n_i)$ is the set of immediate predecessors of task n_i. The algorithm targets scheduling of all critical tasks (i.e., tasks on the critical path of the *DAG*) onto a single processor, in which the critical tasks are executed in minimum time as possible. If the selected task is noncritical, the processor selection phase is based on earliest execution time with insertion-based scheduling. Like HEFT algorithm, the CPOP

algorithm has an $O(n^2p)$ time complexity for n nodes and p processors.

C. Path-based Heuristic Task Scheduling Algorithm

The *PHTS* algorithm is proposed for a bounded number of heterogeneous processors consisting of three phases namely, a path-prioritizing phase, task selection phase, and processor selection phase [25]. Path prioritizing phase for computing the priorities for all possible paths. Each path is assigned by a value called rank(p_j), is given by Equation 5.

$$\boldsymbol{Rank(pj)=\sum_{t_i \epsilon p_j} \bar{w}_i + \overline{c_{i,succ(t_i)}}} \text{----------(5)}$$

Where $\bar{w}_i$ is the average computation cost of a task t_i. It is computed by $\bar{w}_i = \sum_{j=1}^{m} w_{i,j}/m$, and $\overline{c_{i,succ(t_i)}}$ is the communication cost of edge from task t_i to its successor, if exists.

In task selection phase, the algorithm selects the unscheduled tasks from the paths in the sorted path list. During the task selection, the algorithm applies the following conditions on each task:

- The task should not be scheduled earlier.
- The task has no parents or its parents are scheduled.

Finally, the algorithm apply the processor selection phase like HEFT algorithm. The algorithm has an $O(n^2p)$ time complexity for *n* nodes and *p* processors.

D. Highest Communicated Path of Task Algorithm

HCPT algorithm consists of three phases called, level sorting phase, task prioritizing phase and processor selection phase. In level sorting phase, the given DAG is traversed in a top-down fashion to sort tasks in each level. In task prioritizing phase, the HCPT algorithm computes the task priority by using the rank value as shown in Equation 6.

$$\boldsymbol{Rank(t_i)= MCP(t_i)+max_{t_j \in succ(t_i)} \left(\overline{c_{i,j}} + Rank(t_j)\right)} \text{------(6)}$$

Where MCP(t_i) refers to Mean Communication of Parent tasks. It is computed by Equation 7.

$$\boldsymbol{MCP(t_i)=(\sum_{j=1}^{n} C_{j,i})/y} \text{ ----------(7)}$$

Where y is the number of parent tasks. Finally, the algorithm apply the processor selection phase like HEFT and PHTS algorithms [8].

IV. Our Scheduling Algorithm

The **N**ode **D**uplication in **C**ritical **P**ath (*NDCP*) algorithm is developed for static task scheduling algorithm for *HDCS* with limited number of processors. This algorithm based on *Critical Path Merge [26] (CPM)* technique and task duplication technique.

Any algorithm applying the list scheduling technique has the freedom to define the two criteria: the priority scheme for the nodes and the choice criterion for the processor. Fig. 2 and Fig. 3 show steps of *NDCP* algorithm. It consists of two phases namely, priority phase and processor selection phase.

A. Priority phase

In this paper, *NDCP* algorithm modified into priority scheme, where gives the priority to the path instead of the node.

```
Schedule_Task(t_i)
Begin
For each processor P_j in the processor set (P_j ϵ Q) do
    Compute EFT(t_i, P_j) value
  End for
Assign t_i to the p_j that minimizes EFT
If t_i is VIT
{
 If DRT ( t_i , p_j)>w (MP,p_j)
  If EST(t_i ,p_j)>EFT(MP, p_j)
   {
    Duplicate MP on P_j without violate the dependency constraints
    Update EFT of t_i on p_j
   }
}
End
```

Fig. 2. Schedule Task Function

```
Set the computation cost of tasks & communication cost of edges
While there are tasks in given DAG do
  Compute the Critical Path
                using CP_x=Max{∑_1^n w(t_i)_max + ∑ C_(i,succ(t_i))}
  Put critical path in Critical Path List (CPL)
  Remove critical path from the DAG
  Update the DAG
End While
For each path CP_x in CPL
For each task t_i in CP_x
 If t_i has no parents or all parents are scheduled then
   {
    Call Schedule_Task(t_i)
    For each Waited Task t_w in WL
     If all parents of t_w are scheduled then
      {
       Call Schedule_Task(t_w)
       Remove t_w from WL
      }
    End for
   }
 Else
   Put task t_i in Waited List (WL)
 End for
End for
```

Fig. 3. Node Duplication in Critical Path Algorithm (NDCP)

The *NDCP* algorithm computes the critical path of the *DAG* using Equation 8.

$$\boldsymbol{CP=Max\{\sum_1^b w(t_i)_{max} + \sum c_{i,succ(t_i)}\}} \text{ ----------(8)}$$

Where $w(t_i)_{max}$ is the maximum computation of task t_i. b is the number of CP tasks. $\boldsymbol{C_{i,succ(t_i)}}$ is the communication between t_i and its successor, where t_i and its successor belong to the same critical path. Then the algorithm removes this critical path.

After updating the *DAG*, the algorithm computes next critical path and so on. The *NDCP* computes a critical path using the largest weight for each task t_i at slowest processor p_j,. It sorts all critical paths into critical path list *CPL* in descending order.

B. Processor selection phase

This phase consists of two stages: processor stage and duplication test stage. In processor stage, *NDCP* algorithm selects a CP_x from *CPL*, and then it selects task t_i from CP_x. If t_i has no parents or all parents are scheduled, the algorithm calls *Scheduled_Task* function (as shown in Fig. 2). In *Schedule_Task* function, the *NDCP* algorithm calculates *EFT* of task t_i by Equation 2 for each processor, and selects the processor that has a minimum *EFT* to assign the task. With high performance algorithms, some processors are idle during the execution of the application because of DRT. If DRT is enough to duplicate MP, the execution time of the parallel application could be reduced [11]. *So, the algorithm applies task duplication to reduce the makespan. Schedule_Task* function executes also stage of duplication test. The algorithm test, if *DRT* of task t_i is more than the *weight* of MP on the same processor p_j ,the algorithm duplicates the MP on p_j and updates *EFT* of task t_i. The duplication stage is applied on VIT only. This must be done without violating the precedence constrains among tasks. If the task has parents without scheduled, the algorithm puts this task in waiting list W_L to be ready. Once all parents are scheduled, the algorithm selects the task from W_L to schedule. It also removes that task from W_L and continues. Using W_L guarantees scheduling all of important tasks early. A case study is taken into account as following.

Case Study: Considering the application DAG shown in Fig. 4, Table 1 shows the computation matrix. The generated schedule along with stepwise trace of the *HCPT* algorithm and *NDCP* algorithm are shown in Fig. 5. With applying task duplication, DRT of tasks decreases. So the schedule length with task duplication decreases. *PHTS*, *HEFT* and *CPOP* algorithms were also applied on sample DAG 1, and the results respectively were 47,47, 57. From Fig. 5 it is clear that our algorithm is outperforms the others because it marks 43 units. So the scheduling performance is enhanced. It is noted that, the *NDCP* algorithm applies task duplication to decrease the communication overhead by using idle time in scheduling. The NDCP algorithm applies the task duplication on VIT only, because the task that belongs to the critical path is critical task. So, if EFT of VIT decreases the schedule length of application will decrease. When the algorithm duplicates a task, it decreases *DRT* of its childs and decreases also *EFT*. This leads to good utilization of processors in the system.

TABLE I. COMPUTATION MATRIX

Task	P0	P1
T_1	10	7
T_2	8	5
T_3	5	2
T_4	15	10
T_5	12	14
T_6	3	7
T_7	8	4
T_8	4	3
T_9	9	7

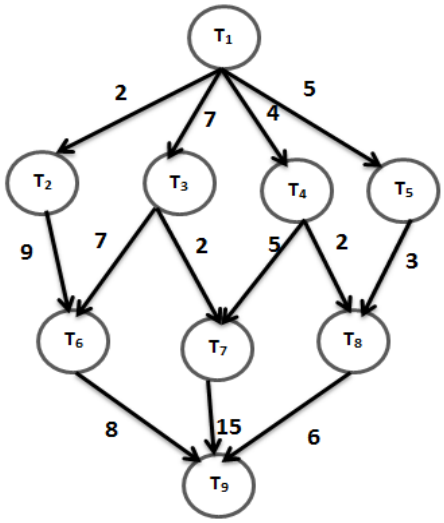


Fig. 4. Sample DAG 1

V. RESULTS AND DISCUSSIONS

A. Simulation Environment

To evaluate the performance of our developed *NDCP* algorithm, a simulator had been built using visual *C# .NET 4.0* on machine with:

- Processor: Intel(*R*) Core(*TM*) *i3 CPU M 350 @2.27GHz*.
- Installed memory *RAM*: *4.00 GB*.
- System type: window 7, 64-bit.

To test the performance of *NDCP* algorithm with the other algorithms a set of randomly generated graphs is created by varying a set of parameters that determines the characteristics of the generated *DAGs*.

These parameters are described as follows:

- *DAG* size: *n*: The number of tasks in the *DAG*.
- Density:

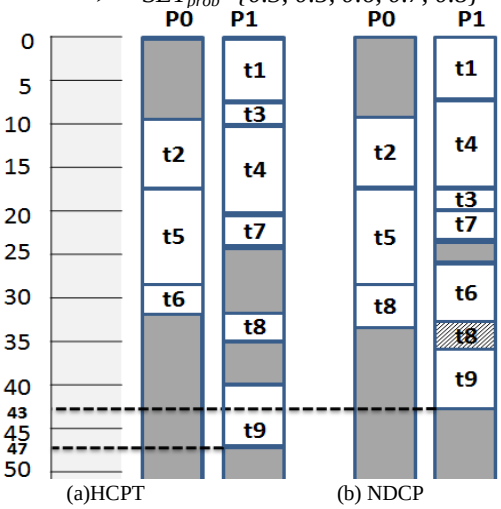
It is used "*sameprob*" method to generate the DAG [27]. Let *A* denote a task connection matrix with elements *a(i,j)*, where $0 \le i \le n$, and $0 \le j \le n$, represent the task number (t_0 is the entry dummy node and t_n is the exit dummy node). When *a(i,j)=1*, t_i precedes task t_j, when *a(i,j)=0*, t_i and t_j are independent of each other. In the "*sameprob*" edge connection method, *a(i,j)* is determined by independent random values defined as follows:

P[a(i,j)=1]<=prob for 1≤i<j≤n and *P[a(i,j)=0]> prob for 1≤i<j≤n, P[a(i,j)=0]=1* if *i≥j*, where *prob* indicates the probability that there exists an edge (precedence constraint) between t_i and t_j .

With six different numbers of processors varying from 8, 16, 32, 64 and 80 processors. For each number of processors, six different *DAG* sizes have been generated varying from 40, 60, 80,100,120 and 150 tasks. In each experiment, the

probability *p* is assigned from the corresponding set given below:

- SET_{prob}={0.3, 0.5, 0.6, 0.7, 0.8}



(a)HCPT (b) NDCP

Fig. 5. The Schedules Generated by Different Algorithms

HCPT is applied *(*which is the pest performance algorithm compared with *HEFT, PHTS, CPOP), PHTS (*which is the pest performance algorithm compared with *HEFT)* and *NDCP* algorithms on Standard Task Graph Set *(STG)* (a kind of benchmark for evaluation of multiprocessor scheduling algorithms) [28]. The results of *PHTS*, HCPT and *NDCP* respectively were 254, 230 and 205 units on random task graph 50//tmp/50/rand0005.stg. In addition, the algorithms are applied on random task graph 50//tmp/50/rand0000.stg, and the results of *PHTS*, *HCPT* and *NDCP* were 88, 80 and 76 units respectively, from the results, it is noted that our algorithm outperforms other algorithms compared in performance.

B. Comparison Metrics and Results

The comparison metrics are schedule length, speedup, efficiency, and time complexity.

1) Schedule Length

Schedule length is the maximum finish time of the exit task in the scheduled *DAG [26]*. The main function of task scheduling is minimizing an application time, so schedule length is the important metric to measure performance of task scheduling algorithm. The *NDCP* algorithm used critical path to detect task priority, because the critical path contains a very important tasks. The *NDCP* algorithm computes the first critical path to get rid the critical tasks then it computes the next critical path (after updating *DAG*) to get rid the next critical tasks and so on. It deals with the *DAG*, after computing a critical path, as a new *DAG* with new critical path. The *NDCP* algorithm uses also task duplication to reduce DRT of the successors, and it could reduce the overall time of application. The algorithm duplicates MP of VIT only. Therefore, the *NDCP* algorithm is more efficient than other algorithms. This appeared from Fig. 6 to Fig. 10. Figures show scheduling length versus number of tasks with varying number of processors 8, 16, 32, 64 and 80. Performance ratio in schedule length is 11%.

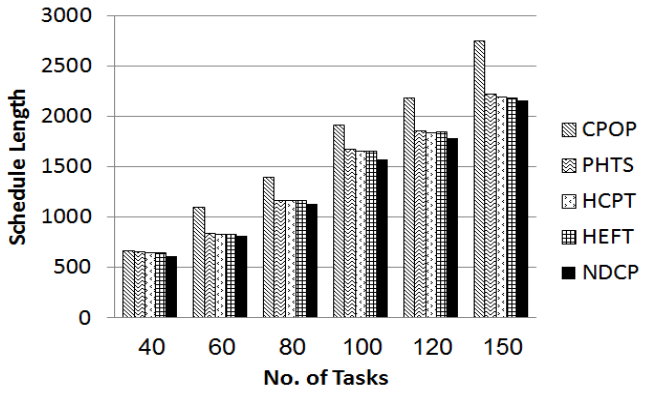


Fig. 6. Schedule Length at 8 Processors

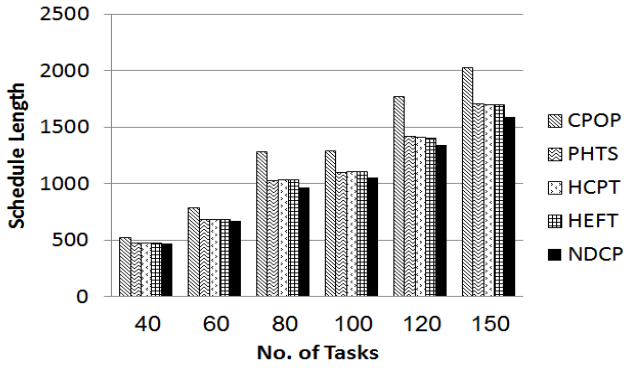


Fig. 7. Schedule Length at 16 Processors

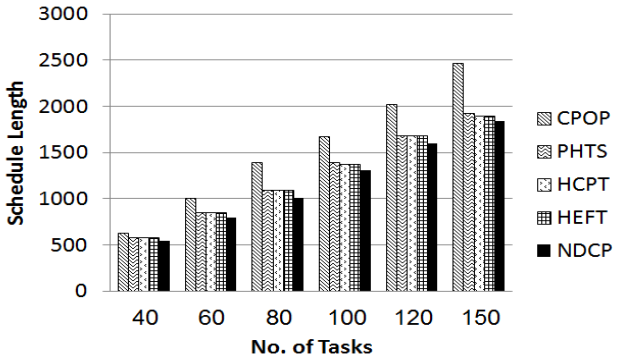


Fig. 8. Schedule Length at 32 Processors

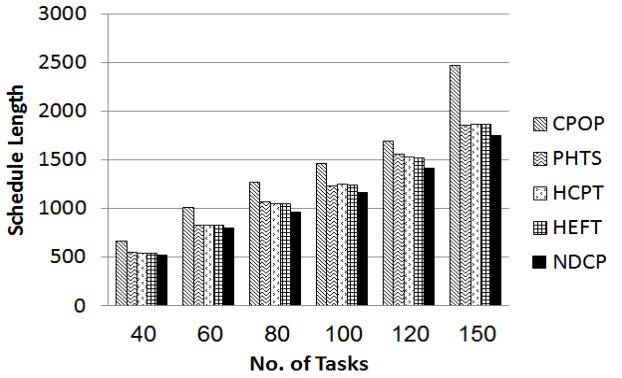


Fig. 9. Schedule Length at 64 Processors

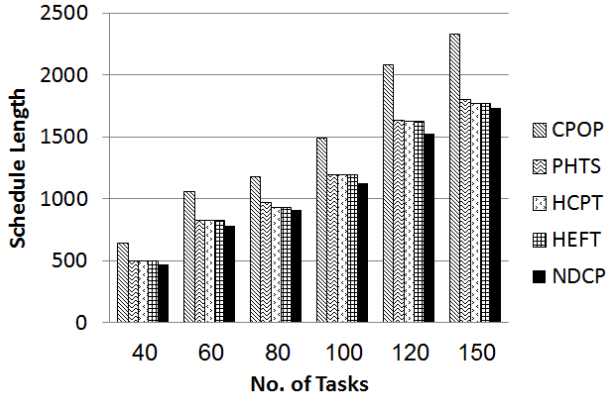


Fig. 10. Schedule Length at 80 Processors

2) Speedup

Speedup of a schedule is defined as the ratio of the schedule length obtained by assigning all tasks to the fastest processor, to the schedule length of application [24]. The speedup is given by Equation 9.

$$\textbf{Speedup}=\frac{\underset{p_j \epsilon P}{Min}[\sum_{ni \epsilon V} w(i,j)]}{SL} \quad \text{---------- (9)}$$

Where $w(i,j)$ means the weight of task t_i on processor p_j and *SL* means the schedule length. Speedup is a good measure for the execution of an application program on a parallel system. Due to minimize schedule length, all processors have finished tasks execution earlier and speedup of *NDCP* algorithm increases. The results of the comparative study according to the speedup parameter have been presented from Fig. 11 to Fig. 16. According to the results, performance ratio of speedup is calculated as 10.5%.

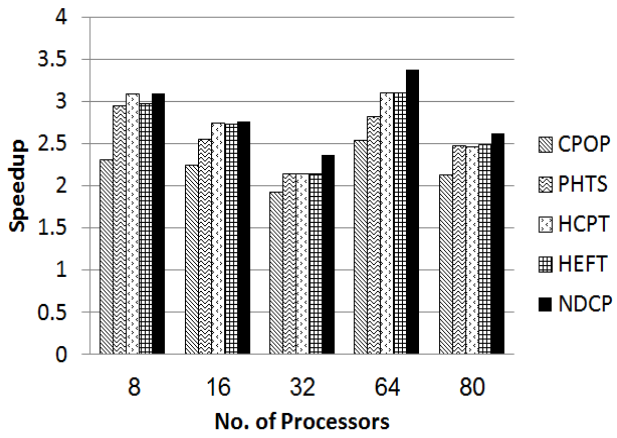


Fig. 11. Speedup with 40 Tasks

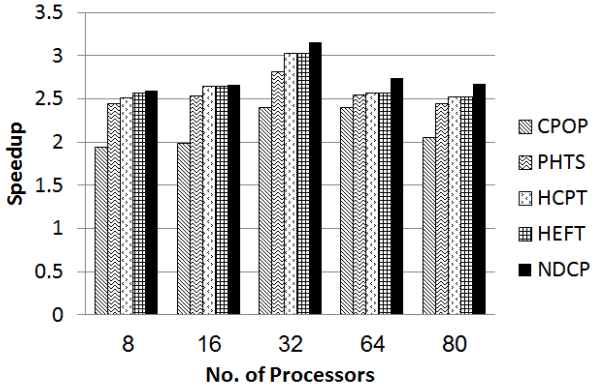


Fig. 12. Speedup with 60 Tasks

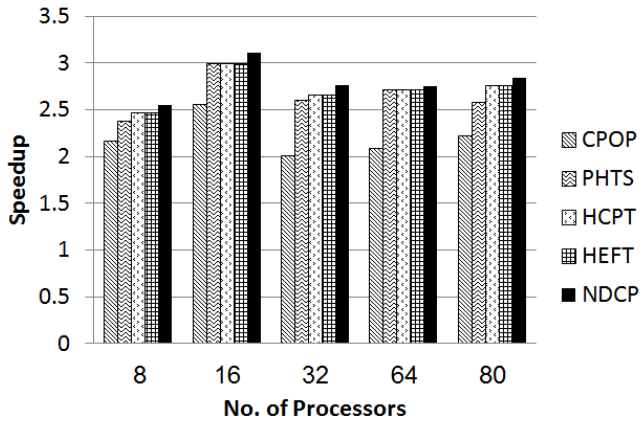


Fig. 13. Speedup with 80 Tasks

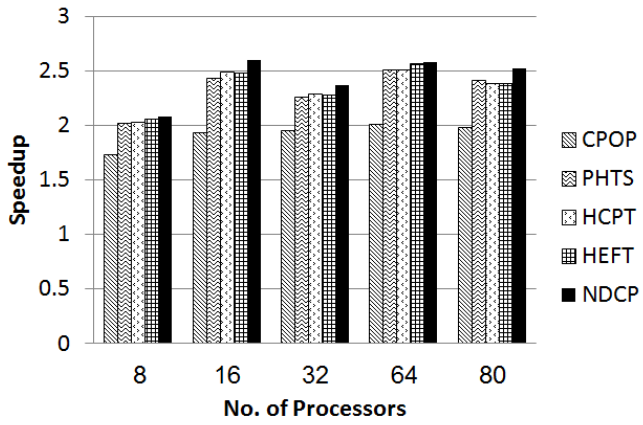


Fig. 14. Speedup with 100 Tasks

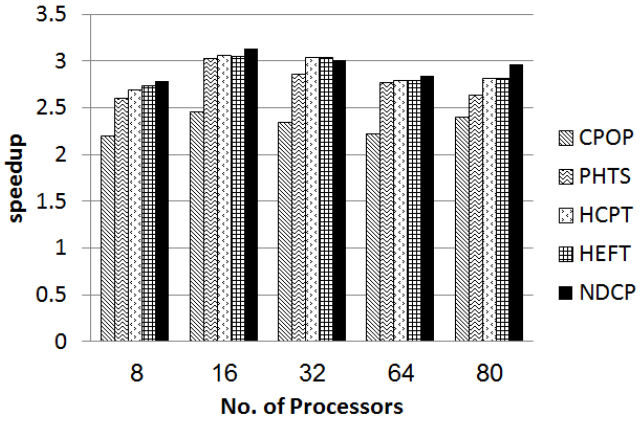


Fig. 15. Speedup with 120 Tasks

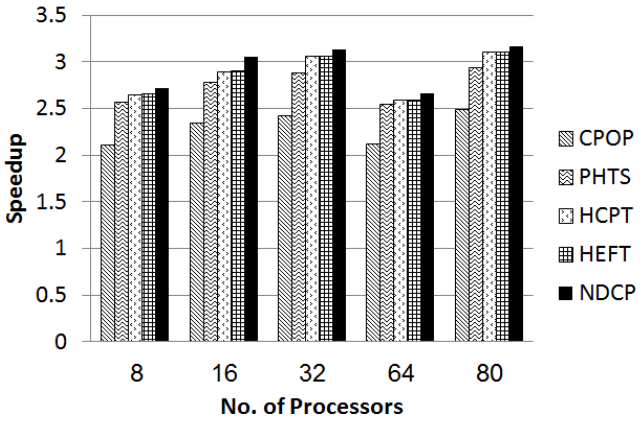


Fig. 16. Speedup with 150 Tasks

3) Efficiency

Efficiency is the speedup divided by the number of processors used [24]. The efficiency is described in Equation 10.

$$Efficiency = \frac{Speedup}{number\ of\ processors\ used} \quad \text{----------(10)}$$

Using task duplication involves the largest number of parallel computers and makes balance between them. Efficiency is an indication to what percentage of a processors time is being spent in useful computation. So efficiency of the *NDCP* algorithm outperforms efficiency of the other algorithms. From Fig. 17 to Fig. 22, figures show efficiency of the *NDCP* algorithm compare with *HEFT*, *CPOP*, *PHTS* and HCPT algorithms. The performance ratio in efficiency which has been achieved by *NDCP* algorithm is 9.3%. According to efficiency parameter, our proposed *NDCP* algorithm achieves better performance than the other algorithms.

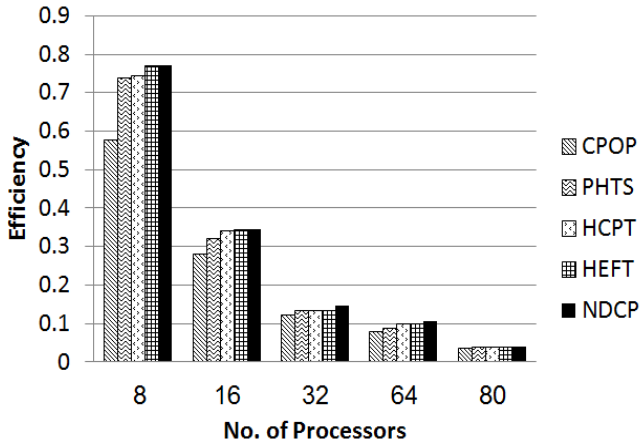


Fig. 17. Efficiency with 40 Tasks

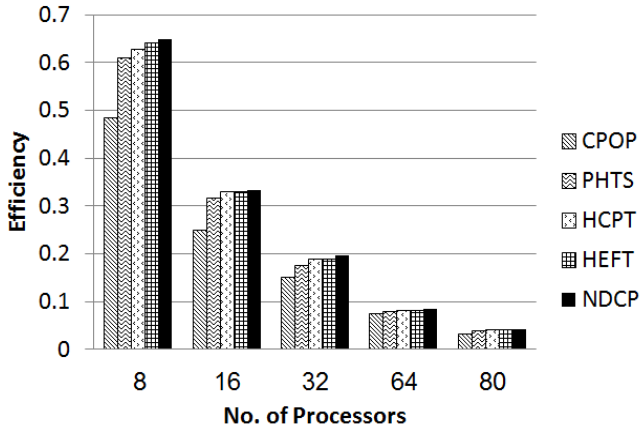


Fig. 18. Efficiency with 60 Tasks

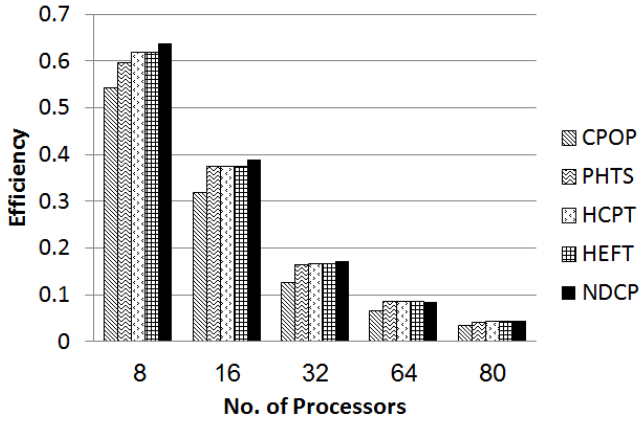


Fig. 19. Efficiency with 80 Tasks

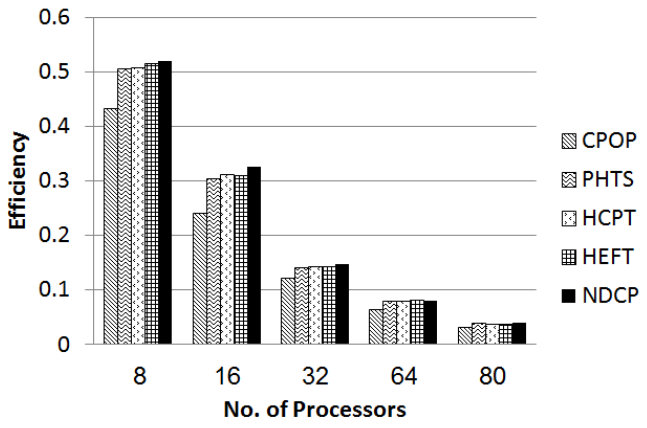


Fig. 20. Efficiency with 100 Tasks

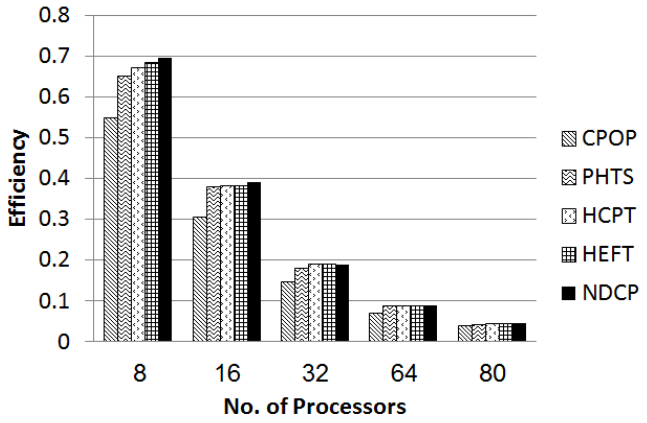


Fig. 21. Efficiency with 120 Tasks

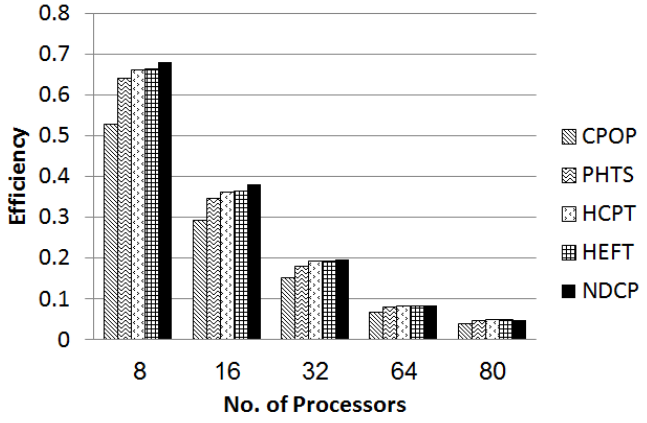


Fig. 22. Efficiency with 150 Tasks

4) Time Complexity

Time complexity is the amount of time taken to assign every task to specific processor according to specific priority. The *NDCP* algorithm has an *O(cp+w*p*n)* time complexity for *cp* number of critical paths, w number of waited tasks w<n, *n* number of tasks and *p* number of processors.

Our algorithm may approximate time complexity into *O(wpn)*. From Table II; it is noted that, task duplication algorithms have high time complexity. But *NDCP* algorithm has the lowest time complexity because The *NDCP* algorithm tests task duplication, if there is an idle time at specific processor not at all processors. The algorithm assigns the task firstly then examines, if there is enough idle time before the task to duplicate its parent or not so, our algorithm has the lowest time complexity for task duplication. This makes the *NDCP* algorithm outperformance the other algorithms.

TABLE II. TIME COMPLEXITY OF SOME ALGORITHMS

Algorithm	Complexity	Use Duplication
HEFT	$O(n^2p)$	No
CPOP	$O(n^2p)$	No
PHTS	$O(n^2p)$	No
HCPT	$O(n^2p)$	No
HCNF	$O(n^2 \log n)$	Yes
NDCP	O(wnp)	Yes

VI. CONCLUSIONS

In this paper, a new scheduling algorithm has been presented for heterogeneous distributed computing systems (*HDCS*) to enhancement scheduling performance. This algorithm based on *Critical Path Merge (CPM)* technique and task duplication technique. The NDCP algorithm duplicate MP for VIT only. The performance analysis showed that the proposed *NDCP* algorithm has better performance than HCPT, *PHTS*, *HEFT* and *CPOP* algorithms. According to the simulation results, it is found that the *NDCP* algorithm is better than the other algorithms in terms of schedule length, speedup and efficiency. The *NDCP* algorithm also has the lowest time complexity *O(wnp)*. Performance improvement ratio in schedule length is 11%, performance improvement ratio in speedup is 10.5% and performance improvement ratio in efficiency is 9.3%. In addition, the algorithms are applied on Standard Task Graph *STG* as a benchmark, and it is observed that *NDCP* algorithm is more efficient than the other algorithms.

REFERENCES

[1] N. Wahid and N. Wafa, "A New DAG Scheduling Algorithm for Heterogeneous Platforms," in 2nd IEEE International Conference on Parallel Distributed and Grid Computing (PDGC). IEEE, pp.114–119, 2012.

[2] H. Yang, P. Lee and C. Chung , "Improving static task scheduling in heterogeneous and homogeneous computing systems," IEEE International Conference on Parallel Processing, ICPP, pp. 45-45, 2007.

[3] H. Izakian, A. Abraham and V. Snasel, "Comparison of heuristics for scheduling independent tasks on heterogeneous distributed environments," in Proc. of the International Joint Conference on Computational Sciences and Optimization, IEEE, vol. 1, pp. 8-12, 2009.

[4] Sheikh, H.F., Ahmad, I. , "Dynamic task graph scheduling on multicore processor for performance, energy, and temperature optimization," IEEE, Published in: Green Computing Conference (IGCC), Page: 1-6, June 2013.

[5] E. Ilavarasan and P. Thambidurai, "Low Complexity Performance Effective Task Scheduling Algorithm for Heterogeneous Computing Environments," Journal of Computer Sciences, 3(2):94-103, 2007.

[6] H. Arabnejad and J. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," IEEE Transactions on Parallel and Distributed Systems, , no. 99, pp. 1–1, 2013.

[7] R. Eswari and S. Nickolas, "Expected completion time-based scheduling algorithm for heterogeneous processors", in Proc. International Conf. Information Communication and Management, IPCSIT vol.16 2011, pp.72-77.

[8] A. A. Nasr, N. A. El-bahnasawy and A. El-sayed, "Task scheduling optimization in heterogeneous distributed systems", International Journal of Computer Applications (0975 – 8887),Volume 107, No 4, December 2014.

[9] A. A. Nasr, N. A. El-bahnasawy and A. El-sayed, "Task scheduling algorithm for high performance heterogeneous distributed systems", International Journal of Computer Applications (0975 – 8887) , Volume 110 – No. 16, January 2015.

[10] H.Topcuoglu, S. Hariri, and M.Y.Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing",IEEE Trans. Parallel and Distributed Systems, Vol. 13, No.3, pp. 260- 274, March 2002.

[11] I. Ahmad, and Y. Kwok, "A New approach to scheduling parallel programs using task duplication," Proc. International Conf. Parallel Processing, Vol.2, pp. 47-51, 1994

[12] J. Mei and K Li, "Energy-Aware scheduling algorithm with duplication on heterogeneous computing systems," Publish in: Grid Computing (GRID), ACM/IEEE 13th International Conference, Page: 122 -129, Sept. 2012.

[13] Baskiyar, S.; SaiRanga, P.C., "Scheduling directed a-cyclic task graphs on heterogeneous network of workstations to minimize schedule length," *2003 International Conference on Parallel Processing Workshops*, pp.97,103, 6-9 Oct. 2003.

[14] S. Darbha and D. P. Agrawal, "A task duplication based scalable scheduling algorithm for distributed memory systems". J. Parallel Distrib. Comput, Vol. 46, PP. 15-27, 1997.

[15] Y. Xu, K. Li, T. T. Khac and M. Qiu, "A multiple priority queuing genetic algorithm for task scheduling on heterogeneous computing systems," IEEE 14th International Conference on High Performance Computing and Communications, pp. 639-646, 2012.

[16] M. Gallet, L. Marchal and F. Vivien, "Efficient scheduling of task graph collections on heterogeneous resources," IPDPS 2009-Proceeding of the IEEE International Parallel and Distributed Processing Symposium, pp.1-11, 2009.

[17] C. Hui, " A high efficient task scheduling algorithm based on heterogeneous multi-core processor", IEEE, Database Technology and Application (DBTA) Pages 1-4, Nov. 2010.

[18] Y. Kang and Y. Lin, "a recursive algorithm for scheduling of tasks in a heterogeneous distributed environment," IEEE, Biomedical Engineering and information (BMEI), Pages 2099-2103, Vol:4, Oct. 2011.

[19] Rahman M., Venugopal S. and Buyya R., "A dynamic critical path algorithm for scheduling scientific workflow applications on global grids," 3rd IEEE International Conference on e-Science and Grid Computing, pp. 35-42, 2007.

[20] Al Na'mneh, R.A., Darabkh, K.A., "A new genetic-based algorithm for scheduling static tasks in homogeneous parallel system," Published in: Robotics, Biomimetics, and Intelligent Computational Systems (ROBIONETICS), 2013 IEEE International Conference, Page: 46-50, Nov. 2013.

[21] Thambidurai P. and Mahilmannan R., "Performance effective task scheduling algorithm for heterogeneous computing system," IEEE Proceedings of the 4th International Symposium on Parallel and Distributed Computing, pp. 28-39, 2005.

[22] R. Bajaj and D. P. Agrawal, "Improving scheduling of tasks in a heterogeneous environment," IEEE Transactions Parallel Distributed System, vol. 15, pp. 107–118, February 2004.

[23] M. Ehsan, M. Sajjad, H. Altaf, N. Muhammad and A. Shoukat, "SDBATS: A novel algorithm for task scheduling in heterogeneous computing systems," IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, pp.43-53, 2013.

[24] N. A. Bahnasawy, M. A. Koutb, M. Mosa and F. Omara, "A new algorithm for static task scheduling for heterogeneous distributed computing systems," African Journal of Mathematics and Computer Science Research Vol. 4(6), pp. 221-234, June 2011.

[25] R. Eswari and S. Nickolas, "Path-based heuristic task scheduling algorithm for heterogeneous distributed computing systems", International Conference on Advances in Recent Technologies in Communication and Computing, 2010.

[26] K. S. Manoj and T. Rajesh, "A survey on scheduling of parallel programs in heterogeneous systems," International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 1, Issue 8, October 2012.

[27] V. A. F. Almeida, I. M. M Vasconcelos, J. N. C. Árabe and D. A. Menascé. " Using Random Task Graphs to Investigate the Potential Benefits of Heterogeneity in Parallel Systems", Proc. Supercomputing '92, pp. 683-691 (1992).

[28] http://www.kasahara.elec.waseda.ac.jp/schedule/index.html.