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, prioritization 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 $_ {i}$ ∈ 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$_ {j}$) ∈ E represents a precedence constraint, such that the execution of t$_ {i}$ cannot be started before t$_ {j}$ finishes its execution. If (t$_ {i}$, t$_ {j}$) ∈ E, then t$_ {i}$ is a parent of t$_ {j}$ and t$_ {j}$ is a child of t$_ {i}$. A task with no parents is called an entry task $t_{\text{entry}}$, and a task with no children is called an exit task $t_{\text{exit}}$. Each edge (t$_ {i}$, t$_ {j}$) ∈ 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 nxm computation cost matrix C stores the execution costs of tasks. Each element c$_{ij}$ ∈ 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.

![DAG and Computation Cost Matrix](image)

**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.

$$EST(t_{i}, P_{j}) = \max\{T_{\text{Available}}(P_{j}), \max\{AFT(t_{k})+c_{jk}\}\}$$

Where $T_{\text{Available}}(P_{j})$ 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 $k=1, 2, ..., n$) on the processor $P_{j}$. $c_{jk}$ is the communication cost from task $t_{k}$ to task $t_{i}$, $c_{jk}$ equal zero if the predecessor task $t_{k}$ is assigned to processor $P_{j}$. For the entry task, $EST(t_{\text{entry}}, P_{j}) = 0$.

**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.

$$EFT(t_{i}, P_{j}) = EST(t_{i}, P_{j}) + w_{ij}$$

Which is the Earliest Start Time of a task $t_{i}$ on a processor $P_{j}$ plus the computational cost $w_{ij}$ of $t_{i}$ on 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.

$$Rank_{u} = \bar{w}_{i} + \max_{n_{j} \in succ(n_{i})}(\bar{c}_{ij} + rank_{d}(n_{j}))$$

Where $succ(n_{i})$ is the set of immediate successors of task $n_{i}$, $\bar{c}_{ij}$ is the average communication cost of edge$(i,j)$, and $\bar{w}_{i}$ 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 $O(n^{2}p)$ 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 upward rank + downward rank value at each step. Downward rank can be calculated by Equation 4.

$$Rank_{d}(n_{i}) = \max_{n_{j} \in pred(n_{i})}(\bar{c}_{ij} + \bar{w}_{j} + rank_{d}(n_{j}))$$

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($p_i$), is given by Equation 5.

$$\text{Rank}(p_i)=\sum_{t \in P} \bar{w}_i + C_{\text{Succ}}(t_i)$$

Where $\bar{w}_i$ is the average computation cost of a task $t_i$. It is computed by $\bar{w}_i = \sum_{j=1}^{m} w_{ij}/m$, and $C_{\text{Succ}}(t_i)$ 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.

$$\text{Rank}(t_i)=\text{MCP}(t_i)+\max_{t_j \in \text{Succ}(t_i)} \left( \bar{c}_{t_j} + \text{Rank}(t_j) \right)$$

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

$$\text{MCP}(t_i)=\frac{\sum_{j=1}^{m} C_{ij} y}{y}$$

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 Node Duplication in Critical Path (NDCP) algorithm is developed for static task scheduling algorithm for HDCC 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)
Begin
   For each processor $P_j$ in the processor set ($P_j \in Q$) 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
End
```

Set the computation cost of tasks & communication cost of edges
While there are tasks in given DAG do
   Compute the Critical Path using $CP_t=\text{Max} \{ \sum_{i=1}^{m} w(t_i)_{\text{max}} + \sum_{i} c_{i\text{Succ}}(t_i) \}$
   Put critical path in Critical Path List (CPL)
   Remove critical path from the DAG
   Update the DAG
End while
For each path $CP_t$ in CPL
   For each task $t_i$ in $CP_t$
      If $t_i$ has no parents or all parents are scheduled then
         Call Schedule_Task($t_i$)
      For each Waited Task $t_u$ in WL
         If all parents of $t_u$ are scheduled then
            Call Schedule_Task($t_u$
            Remove $t_u$ from WL
         End for
      Else
         Put task $t_i$ in Waited List (WL)
      End for
   End for
End for
```

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

$$CP_{\text{max}}=\max_{i=1}^{b} w(t_i)_{\text{max}} + \sum_{i} c_{i\text{Succ}}(t_i)$$

Where $w(t_i)_{\text{max}}$ is the maximum computation of task $t_i$. $b$ is the number of CP tasks. $c_{i\text{Succ}}(t_i)$ 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 \( t_i \) as slowest processor \( p_i \). 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, from \( CPL \), and then it selects task \( t_i \) from CP. 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 \( p_i \), the algorithm duplicates the MP on \( p_i \) 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**

<table>
<thead>
<tr>
<th>Task</th>
<th>P0</th>
<th>P1</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>T1</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>T2</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>T3</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>T4</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>T5</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>T6</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>T7</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>
probability $p$ is assigned from the corresponding set given below:

$$\text{SET}_{\text{prob}} = \{0.3, 0.5, 0.6, 0.7, 0.8\}$$

HCPT is applied (which is the best performance algorithm compared with HEFT, PHTS, CPOP, PHTS (which is the best 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%.
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.

\[
\text{Speedup} = \frac{\sum_{i,j} w(i,j)}{SL} \quad (9)
\]

Where \(w(i,j)\) means the weight of task \(i\) 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%.
3) Efficiency

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

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

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.
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.

**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 $O(wpn)$. 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.

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**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complexity</th>
<th>Use Duplication</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEFT</td>
<td>$O(n^2/p)$</td>
<td>No</td>
</tr>
<tr>
<td>CPOP</td>
<td>$O(n^2/p)$</td>
<td>No</td>
</tr>
<tr>
<td>PHTS</td>
<td>$O(n^2/p)$</td>
<td>No</td>
</tr>
<tr>
<td>HCPT</td>
<td>$O(n^2/p)$</td>
<td>No</td>
</tr>
<tr>
<td>HCNF</td>
<td>$O(n^2 \log n)$</td>
<td>Yes</td>
</tr>
<tr>
<td>NDCP</td>
<td>$O(wpn)$</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**REFERENCES**


