A Leveled Dag Critical Task First Schedule Algorithm in Distributed Computing Systems

Amal EL-NATTAT  
Computer Science & Eng. Dept.,  
Faculty of Electronic Eng.  
Menouf 32952, Egypt

Nirmeen A. El-Bahnasawy  
Computer Science & Eng. Dept.,  
Faculty of Electronic Eng.  
Menouf 32952, Egypt

Ayman EL-SAYED, IEEE Senior Mem.  
Computer Science & Eng. Dept  
Faculty of Electronic Eng.  
Menouf 32952, Egypt

Abstract—In distributed computing environment, efficient task scheduling is essential to obtain high performance. A vital role of designing and development of task scheduling algorithms is to achieve better makespan. Several task scheduling algorithms have been developed for homogeneous and heterogeneous distributed computing systems. In this paper, a new static task scheduling algorithm is proposed namely; Leveled DAG Critical Task First (LDCTF) that optimizes the performance of Leveled DAG Prioritized Task (LDPT) algorithm to efficiently schedule tasks on homogeneous distributed computing systems. LDPT was compared to B-level algorithm which is the most famous algorithm in homogeneous distributed systems and it provided better results. LDCTF is a list based scheduling algorithm which depends on sorting tasks into a list according to their priority then scheduling one by one on the suitable processor. LDCTF aims to improve the performance of the system by minimizing the schedule length than LDPT and B-level algorithms.

Keywords—Task scheduling; Homogeneous distributed computing systems; Precedence constrained parallel applications; Directed Acyclic Graph; Critical path

I. INTRODUCTION

Distributed systems have emerged as powerful platforms for executing parallel applications. A distributed system can be defined as a collection of computing systems that appears to its users as a single system, these systems collaborate over a network to achieve a common goal [1]. There are two types of distributed systems; homogeneous (in which processors are identical in capabilities and functionality) and heterogeneous (in which processors are different).

In distributed computing environment, an application is usually decomposed into several independent and/or interdependent sets of cooperating tasks. Dependent tasks are represented by a Directed Acyclic Graph (DAG). DAG can be defined as a graph consists of a set of vertices or nodes and a set of edges G(V, E) in which each node represents a task and each edge represents a communication between two tasks (the two tasks are dependent on each other). The computation cost of the task is represented by a weight associated with each node and the communication cost between two tasks is represented by a weight associated with each edge. The communication cost between two dependent tasks is considered to equal zero if they are executed on the same processor. Figure 1 shows an example of a simple task graph (DAG). In the Figure, t0 is called predecessor (or parent) of t2 and t2 is called successor (or child) of t0. The edge between t0 and t2 means that t2 can start execution only after t0 finishes its execution. Efficient task scheduling of application tasks is essential to achieve high performance in parallel and distributed systems. The basic function of task scheduling is to determine the allocation of tasks to processors and their execution order in order to satisfy the precedence requirements and obtain minimum schedule length (or makespan) [2]. Task-scheduling algorithms are broadly classified into two basic classes: static and dynamic. In static scheduling, the characteristics of an application, such as execution time of tasks and data dependencies between tasks are known in advance (during compile time before running the application). In dynamic scheduling, some information about tasks and their relations may be undeterminable until runtime [3].

Fig. 1. Example of a DAG

Over the past few decades, researchers have focused on designing task scheduling algorithms for homogeneous and heterogeneous systems with the objective of reducing the overall execution time of the tasks. Topcuoglu et al. [2] have presented HEFT and CPOP scheduling algorithms for heterogenous processors. Luiz et al. [4] have developed lookahead-HEFT algorithm, which look ahead in the schedule to make scheduling decisions. Eswari, R. and Nickolas, S. [5] have proposed PHTS algorithm to efficiently schedule tasks on the heterogeneous distributed computing systems. Rajak and Ranjith [6] have presented a queue based scheduling algorithm called TSB to schedule tasks on homogeneous parallel multiprocessor system. Ahmed, S.G.; Munir, E.U.; and Nisar, W. [7] have developed genetic algorithm called PEGA that provide low time complexity than standard genetic algorithm (SGA). Xiaoyong Tang; Kenli Li; Renfa Li; and Guiping Liao [8] have presented a list-scheduling algorithm called HEFD for...
heterogeneous computing systems. Nasri, W. and Nafti, W. [9] have developed a new DAG scheduling algorithm for heterogeneous systems that provide better performance than some well-known existing task scheduling algorithms.

In homogeneous distributed systems, researchers have developed many heuristic task-scheduling algorithms such as ISH [10], ETF [11], DLS [12], MH [13], B-level [14] and some heuristics that depend on the critical path such as MCP [15], FCP [16], and CNPT [17]. Among these algorithms, B-level provides the best performance in terms of schedule length, speedup, and efficiency. LDPT (Leveled DAG Prioritized Task) algorithm [18] was compared to B-level algorithm which is the most famous algorithm in homogeneous distributed systems and it provided better results.

In this paper, the problem of scheduling precedence constrained parallel tasks on homogeneous physical machines (PMs) is addressed. A new static scheduling algorithm called LDCTF is proposed. The goal of LDCTF is to optimize the performance of LDPT [18] algorithm in order to provide better system performance. LDCTF is a list scheduling algorithm. It depends on dividing the DAG into levels then sorting tasks in each level into a list according to their priority and finally, picking tasks from the list one by one to schedule it on the suitable processor. LDCTF is compared to LDPT and B-level algorithms and it provided better results in terms of schedule length, speedup, and efficiency.

The remainder of this paper is organized as follows. Section II provides an overview of the related work algorithm. The proposed algorithm is discussed in section III. Section IV presents performance evaluation results of the proposed algorithm. Finally, conclusion and future work is reviewed in section V.

II. LDPT ALGORITHM

LDPT is a list based scheduling algorithm. It depends on dividing the DAG into levels with considering the dependency conditions among tasks in the DAG. The algorithm has two phases: (1) Task prioritization phase, (2) Processor selection phase. LDPT algorithm depends on giving a priority to each task as shown in Figure 2 then; scheduling each task on one processor with taking into consideration the insertion-based policy. Figure 2 shows the pseudo code of LDPT algorithm.

III. LEVELED DAG CRITICAL TASK FIRST (LDCTF) ALGORITHM

LDCTF is a theoretical task scheduling algorithm. LDCTF, LDPT, and B-level algorithms are applied on Standard Task Graph STG [19] as a bench mark, and it was found that LDCTF algorithm is more efficient than LDPT and B-level algorithms.

LDCTF is a list based scheduling algorithm. It depends on dividing the DAG into levels with considering the dependency conditions among tasks in the DAG then, applying the Min-min method [20] which means calculating the minimum completion time (MCT) for each task on all processors then selecting the task with the lowest MCT to schedule. The algorithm has two phases: (1) Task prioritization phase, (2) Processor selection phase.

Figure 2: LDPT algorithm [18]

A. Task prioritization phase:

In this phase, the critical path [2] is calculated for the DAG (critical path is the longest path from the entry task to the exit task in the graph) then, the DAG is divided into levels and the tasks in each level will be sorted into a list based on their priority. The priority for each task is given as follow:

1) First, the critical task (task located on the critical path) in each level will have the highest priority.
2) Then, the expected Earliest Finish Time (EFT) is calculated for the other tasks in the same level and the task with the lowest EFT will have the highest priority. If tow tasks have equal EFT value then, the task with the lowest task number will have the highest priority. EFT of a task tion processor Pj is computed as follow:

\[
EFT(t_i, P_j) = \frac{w_{i, j}}{P_j} + EST(t_i, P_j) = \frac{w_{i, j}}{P_j} + EST(t_i, P_j) - \text{Start Time of the selected task}
\]

3) Finally, tasks in each level are sorted into the list in ascending order according to their EFT value.

B. Processor Selection Phase:

In this phase, the tasks are picked from the list one by one and assigned to the processor that will minimize the earliest start time of the task, with taking into consideration the insertion-based policy. The insertion policy means that if there is an idle time slot on the processor between two already scheduled tasks and it was enough for executing the task, then the task is assigned on that processor in this idle slot without violating precedence constraints. In other words, a task can be scheduled earlier if there is a period of time between two tasks already scheduled on processor (P), where P runs idle. If two processors provide the same start time for the task then, the task is assigned to the first processor that will minimize the EST of it.

The Earliest Start Time of a task \( n_i \) on a processor \( P_j \) is defined as:

\[
EST(n_i, P_j) = \min\{EST(n, P_j) : \forall n \in \text{precede } n_i \}
\]
$EST(n_x, P_m) = \max[TAvailable(P_m), \max\{AFT(n_i) + c_{x,i}\}]$ (2)

Where $TAvailable(P_m)$ is the earliest time at which processor $P_m$ is ready. $AFT(n_i)$ is the Actual Finish Time of a task $n_i$ (the parent of task $n_x$) on the processor $P_m$. $c_{n,i}$ is the communication cost from task $n_i$ to task $n_x$. $c_{k,i}$ equal zero if the predecessor task $t_k$ is assigned to processor $P_m$. For the entry task, $EST(n_{entry}, P_m) = 0$. Figure 3 shows the pseudo code of LDCTF algorithm.

### C. Case Study

Consider the DAG shown in Figure 4; assume the system has two processors ($P_0$, $P_1$). The critical path for the DAG in Figure 4 is $(t_0, t_1, t_3, t_6, t_8)$. Table 1 shows the computation cost for each task. Both LDPT and LDCTF algorithms generate a list of tasks that shows the execution order of them. Table 2 shows the lists generated by LDPT and LDCTF algorithms. For LDCTF algorithm the critical task in each level will be scheduled first as shown in Table 2. Figure (5.a, 5.b) shows the Gantt chart generated by LDPT and LDCTF algorithms respectively. Both algorithms assign the selected task to the processor that minimizes the start time (EST) of it. For example, in Figure 5.a, the EST for task $t_2$ on $p_0$ is 5 and the EST for $t_2$ on $p_1$ is 4, so the task $t_2$ is scheduled on $p_1$. In Figure 5.b, the same manner if followed with taking into consideration the insertion-based policy. From Figure 5, it is shown that the schedule length (the finish time of the last task scheduled from the DAG) resulted from LDPT and LDCTF algorithms is 25, and 23 unit of time respectively.

-- Table I.

<table>
<thead>
<tr>
<th>Task</th>
<th>Computation Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>2</td>
</tr>
<tr>
<td>$t_1$</td>
<td>3</td>
</tr>
<tr>
<td>$t_2$</td>
<td>1</td>
</tr>
<tr>
<td>$t_3$</td>
<td>4</td>
</tr>
<tr>
<td>$t_4$</td>
<td>3</td>
</tr>
<tr>
<td>$t_5$</td>
<td>5</td>
</tr>
<tr>
<td>$t_6$</td>
<td>2</td>
</tr>
<tr>
<td>$t_7$</td>
<td>4</td>
</tr>
<tr>
<td>$t_8$</td>
<td>6</td>
</tr>
</tbody>
</table>

Figure 5 depicts the Gantt chart generated by LDPT and LDCTF algorithms. From the Figure, it is shown that the schedule length generated from LDPT algorithms is 25 unit time while the schedule length generated from LDCTF algorithm is 23 unit time. In case of LDCTF, we observe that there is less periods in which processors are idle than LDPT. According to this result, the overall running time of the application will be decreased and the efficiency of the system will be improved.

-- Table II.

<table>
<thead>
<tr>
<th>Execution</th>
<th>LDPT</th>
<th>LDCTF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$t_0$</td>
<td>$t_0$</td>
</tr>
<tr>
<td>2</td>
<td>$t_1$</td>
<td>$t_1$</td>
</tr>
<tr>
<td>3</td>
<td>$t_2$</td>
<td>$t_2$</td>
</tr>
<tr>
<td>4</td>
<td>$t_3$</td>
<td>$t_3$</td>
</tr>
<tr>
<td>5</td>
<td>$t_4$</td>
<td>$t_4$</td>
</tr>
<tr>
<td>6</td>
<td>$t_5$</td>
<td>$t_5$</td>
</tr>
<tr>
<td>7</td>
<td>$t_6$</td>
<td>$t_6$</td>
</tr>
<tr>
<td>8</td>
<td>$t_7$</td>
<td>$t_7$</td>
</tr>
<tr>
<td>9</td>
<td>$t_8$</td>
<td>$t_8$</td>
</tr>
</tbody>
</table>
IV. RESULTS AND PERFORMANCE EVALUATION

A. Simulation Environment

To evaluate the performance of LDCTF algorithm, a simulator had been built using visual C# .NET 4.0 on machine with: Intel(R) Core(TM) i3 CPU M 350 @2.27GHz, RAM of 4.00 GB, and the operating system is window 7, 64-bit. To test the performance of LDPT and LDCTF 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 DAG). Density (d: the probability of existence edge between ni in levelj and nx in the next level levelj+1 for DAG. Where, i, x=1,2,...., N, and N is the number of tasks, j=1, 2,..., T, and T is the number of levels in DAG). With six different numbers of processors varying from 2, 4, 8, 16, 32 and 64 processors. For each number of processors, six different DAG sizes have been used varying from 10, 20, 40, 60, 80 and 100 nodes.

B. Evaluation Metrics

The most important metrics for evaluating performance of scheduling algorithms are schedule length, speedup, and efficiency. Schedule length is the maximum finish time of the last task (exit task) scheduled from the DAG.

\[
\text{Schedule length} = \text{Max}(AFT(\text{next})) \tag{3}
\]

Where AFT(\text{next}) is the actual finish time of the exit task. Speedup is defined as the ratio of the schedule length generated from executing the application on one processor to the schedule length generated from executing the application on multiple parallel processors.

\[
\text{Speed up} = \frac{\text{Min}_{\text{Processors}} w(i,j)}{SL} \tag{4}
\]

Where \(w(i,j)\) means the weight of task ni on processor pj and SL means the schedule length. Efficiency is the inverse of speedup.

\[
\text{Efficiency} = \frac{\text{speedup}}{\text{Number of processors}} \tag{5}
\]

C. Experimental Results

The schedule length generated by LDPT and LDCTF algorithms is shown in Figure 6, 7, 8, 9, 10, 11 for 10, 20, 40, 60, 80, 100 tasks respectively and the results are recorded in table 3. According to the results, the schedule length is decreased that will minimize the running time of the application. The improvement ratio in schedule length is (2.75%). Figure 12, 13, 14, 15, 16, 17 show a comparative study of the speed up of LDPT and LDCTF algorithms in case of 2, 4, 8, 16, 32, 64 processors respectively. Table 4 shows the speedup results of LDPT and LDCTF algorithms. From the results, we can see that the improving ratio in speed up is (3.2%). Table 5 shows the efficiency results of LDPT and LDCTF algorithms. From Figure 18, 19, 20, 21, 22, 23 we can see that LDCTF is more efficient than LDPT algorithms with improving ratio (1.9%). The schedule length generated by B-level, LDPT, and LDCTF algorithms is shown in Figure 24, 25, 26, 27, 28, 29. Figure 30, 31, 32, 33, 34, 35 shows a comparative study of the speed up of B-level, LDPT, and LDCTF algorithms. The efficiency results of B-level, LDPT, and LDCTF algorithms are shown in Figure 36, 37, 38, 39, 40, 41.
Fig. 8. Schedule length for 40 task

Fig. 9. Schedule length for 60 task

Fig. 10. Schedule length for 80 task

Fig. 11. Schedule length for 100 task

Fig. 12. Speedup on 2 processors

Fig. 13. Speedup on 4 processors

Fig. 14. Speedup on 8 processors

Fig. 15. Speedup on 16 processors
TABLE III. SCHEDULE LENGTH RESULTED FROM LDPT AND LDCTF ALGORITHMS

<table>
<thead>
<tr>
<th>Number of tasks</th>
<th>2 processor</th>
<th>4 processor</th>
<th>8 processor</th>
<th>16 processor</th>
<th>32 processor</th>
<th>64 processor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LDPT</td>
<td>LDCTF</td>
<td>LDPT</td>
<td>LDCTF</td>
<td>LDPT</td>
<td>LDCTF</td>
</tr>
<tr>
<td>10</td>
<td>598</td>
<td>619</td>
<td>438</td>
<td>481</td>
<td>503</td>
<td>501</td>
</tr>
<tr>
<td>20</td>
<td>1070</td>
<td>1091</td>
<td>784</td>
<td>763</td>
<td>615</td>
<td>629</td>
</tr>
<tr>
<td>40</td>
<td>2319</td>
<td>2265</td>
<td>1304</td>
<td>1236</td>
<td>827</td>
<td>818</td>
</tr>
<tr>
<td>60</td>
<td>3427</td>
<td>3430</td>
<td>2141</td>
<td>2055</td>
<td>1114</td>
<td>1053</td>
</tr>
<tr>
<td>80</td>
<td>4408</td>
<td>4204</td>
<td>2403</td>
<td>2349</td>
<td>1330</td>
<td>1261</td>
</tr>
<tr>
<td>100</td>
<td>5734</td>
<td>5551</td>
<td>2821</td>
<td>2724</td>
<td>1654</td>
<td>1604</td>
</tr>
</tbody>
</table>

TABLE IV. SPEEDUP RESULTED FROM LDPT AND LDCTF ALGORITHMS

<table>
<thead>
<tr>
<th>Number of processors</th>
<th>10 tasks</th>
<th>20 tasks</th>
<th>40 tasks</th>
<th>60 tasks</th>
<th>80 tasks</th>
<th>100 tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LDPT</td>
<td>LDCTF</td>
<td>LDPT</td>
<td>LDCTF</td>
<td>LDPT</td>
<td>LDCTF</td>
</tr>
<tr>
<td>2</td>
<td>1.5971</td>
<td>1.6040</td>
<td>1.6469</td>
<td>1.7102</td>
<td>1.8893</td>
<td>1.8840</td>
</tr>
<tr>
<td>4</td>
<td>1.9265</td>
<td>2.3298</td>
<td>2.6259</td>
<td>2.7224</td>
<td>3.0162</td>
<td>3.0592</td>
</tr>
<tr>
<td>8</td>
<td>2.3397</td>
<td>2.4219</td>
<td>3.4263</td>
<td>4.0710</td>
<td>4.2586</td>
<td>4.7182</td>
</tr>
<tr>
<td>16</td>
<td>1.9653</td>
<td>2.0157</td>
<td>3.1266</td>
<td>3.7123</td>
<td>5.5702</td>
<td>5.7496</td>
</tr>
<tr>
<td>32</td>
<td>1.9653</td>
<td>2.0157</td>
<td>4.3638</td>
<td>4.4021</td>
<td>4.7180</td>
<td>4.8926</td>
</tr>
<tr>
<td>64</td>
<td>2.0669</td>
<td>2.0881</td>
<td>2.7823</td>
<td>4.6014</td>
<td>4.7749</td>
<td>5.3621</td>
</tr>
</tbody>
</table>

TABLE V. EFFICIENCY RESULTED FROM LDPT AND LDCTF ALGORITHMS

<table>
<thead>
<tr>
<th>Number of processors</th>
<th>10 tasks</th>
<th>20 tasks</th>
<th>40 tasks</th>
<th>60 tasks</th>
<th>80 tasks</th>
<th>100 tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LDPT</td>
<td>LDCTF</td>
<td>LDPT</td>
<td>LDCTF</td>
<td>LDPT</td>
<td>LDCTF</td>
</tr>
<tr>
<td>2</td>
<td>61.596</td>
<td>63.943</td>
<td>55.795</td>
<td>56.945</td>
<td>52.517</td>
<td>53.225</td>
</tr>
<tr>
<td>4</td>
<td>38.207</td>
<td>40.467</td>
<td>34.657</td>
<td>34.955</td>
<td>30.534</td>
<td>30.586</td>
</tr>
<tr>
<td>8</td>
<td>40.216</td>
<td>40.216</td>
<td>23.462</td>
<td>23.431</td>
<td>17.451</td>
<td>17.845</td>
</tr>
<tr>
<td>64</td>
<td>37.264</td>
<td>37.264</td>
<td>18.537</td>
<td>18.537</td>
<td>8.3720</td>
<td>8.3720</td>
</tr>
</tbody>
</table>

Fig. 16. Speedup on 32 processors

Fig. 17. Speedup on 64 processors
Figure 12, 13, 14, 15, 16, 17 depict speedup versus number of processors with varying number of tasks (20, 40, 60, 80, 100). It is shown that LDCTF algorithm provides better speed up than LDPT algorithm. This is because in case of LDCTF algorithm, all processors have finished the execution of tasks earlier than LDPT algorithm.

Figure 18, 19, 20, 21, 22, 23 depict efficiency versus number of processors with varying number of tasks (20, 40, 60, 80, 100). It is shown that LDCTF algorithm is more efficient and provides better performance than LDPT algorithm. Most of processor elements have been perfect utilized in our algorithm because of the communication among tasks is not affected in algorithm breadth procedures.

Figure 24, 25, 26, 27, 28, 29 depicts the schedule length versus number of tasks with varying number of processors 2, 4, 8, 16, 32, and 64 processors. It is shown that the schedule length in case of applying LDCTF algorithm is less than LDPT and B-level algorithms.

Figure 30, 31, 32, 33, 34, 35 depicts speedup versus number of processors with varying number of tasks (10, 20, 40, 60, 80, 100). It is shown that LDCTF algorithm provides better speed up than LDPT and B-level algorithms. This is because in case of LDCTF algorithm, all processors have finished the execution of tasks earlier than LDPT and B-level algorithms.
Fig. 24. Schedule length for 10 tasks

Fig. 25. Schedule length for 20 tasks

Fig. 26. Schedule length for 40 tasks

Fig. 27. Schedule length for 60 tasks

Fig. 28. Schedule length for 80 tasks

Fig. 29. Schedule length for 100 tasks

Fig. 30. Speedup on 2 processors

Fig. 31. Speedup on 4 processors
Fig. 32. Speedup on 8 processors

Fig. 33. Speedup on 16 processors

Fig. 34. Speedup on 32 processors

Fig. 35. Speedup on 64 processors

Fig. 36. Efficiency on 2 processors

Fig. 37. Efficiency on 4 processors

Fig. 38. Efficiency on 8 processors

Fig. 39. Efficiency on 16 processors
Figure 36, 37, 38, 39, 40, 41 depicts efficiency versus number of processors with varying number of tasks (20, 40, 60, 80, 100). It is shown that LDCTF algorithm is more efficient and provides better performance than LDPT and B-level algorithms.

D. Discussion of Results

First, LDCTF algorithm is compared to LDPT algorithm and it provided better results in terms of schedule length, speed up, and efficiency. This is because in case of LDCTF, the critical path is taken into account and the critical task will be scheduled first in each level. This means that the task with the highest computation and communication cost will be scheduled first resulting in minimum schedule length, higher speed up, and higher efficiency.

Finally, LDCTF is compared to B-level algorithm and it provided better results in terms of schedule length, speed up, and efficiency. This is because B-level algorithm depends on paths idea and this will increase the communication overhead during assigning tasks on processors. On the other side, LDCTF algorithm depends on levels idea that will minimize the communication overhead during assigning tasks on processors. Another reason is that B-level algorithm must calculate the b-level value for each task before scheduling so that, the arithmetic calculation in LDCTF is less than B-level algorithm which leads to minimize the complexity factor.

V. Conclusion and Future Work

In this paper, a new static scheduling algorithm (LDCTF) is developed for homogeneous distributed computing systems. The performance of LDCTF algorithm is compared with LDPT algorithm. LDCTF is evaluated for different DAGs and found to be giving better results than LDPT algorithm in terms of schedule length, speed up, and efficiency. LDCTF, LDPT, and B-level algorithms are applied on Standard Task Graph STG as a bench mark, and it was found that LDCTF algorithm is more efficient than LDPT and B-level algorithms. The future scope of the idea can be as follows:

- In this paper LDCTF algorithm is applied on Directed Acyclic Graph (DAG). In the future it can be applied on Directed Cyclic Graph (DCG).
- LDCTF can be applied on Heterogeneous Distributed Computing Systems (HDCS).
- LDCTF can be applied in a dynamic strategy instead of static strategy.
- Finally, duplication technique can be applied with LDCTF algorithm to minimize the communication overhead.

References


