Task Scheduling Algorithm for High Performance Heterogeneous Distributed Computing Systems

Aida A. Nasr
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
Computer Science & Eng.
Dept., Faculty of Electronic Eng.
Menouf 32952, Egypt

ABSTRACT
The main objective of task scheduling is to assign tasks onto available processors with the aim of producing minimum schedule length and without violating the precedence constraints. Several algorithms have been proposed for solving task-scheduling problem. The most of them don't take into account the average communication of parents and data ready time. In this paper, a new static scheduling algorithm is proposed called Communication Leveled DAG with Duplication (CLDD) algorithm to efficiently schedule tasks on the heterogeneous distributed computing systems. It solves most limitations of existing algorithms. The algorithm not only focuses on reducing the makespan, but also provides better performance than the other algorithms in terms of speedup, efficiency and time complexity. It consists of three phases: level sorting phase, task-prioritizing phase and processor selection phase. We evaluate the performance of our algorithm by applying it on random DAGs. According to the evolved results, it has been found that our algorithm outperform the others.

Keywords
Static task scheduling, heterogeneous distributed computing systems, heuristic algorithm.

1. INTRODUCTION
Diverse portions of an application task often require different types of computation. In general, it is impossible for a single machine architecture with its associated compiler, operating systems and programming tools to satisfy all the computational requirements in such an application equally well. Recent developments in high-speed digital communication have made it possible to connect a distributed suite of different high performance machines in order provide a powerful computing platform called Heterogeneous Distributed Computing System (HeDCS). This platform is utilized to execute computationally intensive applications that have diverse computing requirements. However, the performance of parallel applications on such systems is highly dependent on the scheduling of the application tasks onto these machines [1, 2].

Task scheduling [3, 4] is of vital importance in HeDCS 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 the 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 [5, 6, 7]. 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 [8].

In this paper, we focus on static scheduling. Static scheduling is classified into list-based, clustering and duplication based. List scheduling consists of two phases: a task prioritization phase, where a certain priority is computed and is assigned to each node of the DAG, and a machine assignment phase, where each task (in order of its priority) is assigned to machine that minimizes a suitable cost function. List-scheduling is generally accepted as an attractive approach since it pairs low complexity and good results [9, 10]. Examples of list-based algorithms are: Heterogeneous Earliest Finish Time (HEFT) and Critical Path On Processor (CPOP) [11].

Another static scheduling category is task duplication based algorithms [12, 13], in which tasks have been 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. Examples of duplication algorithms are Scalable Task Duplication Based Scheduling (STDS) [14] and Heterogeneous Critical Node First (HCNF) [15].

In this paper, a new algorithm called Communication Leveled DAG with Duplication (CLDD) is developed for static task scheduling for the HeDCS with limited number of processors. The developed algorithm is based on assigning a priority for each node by using Rank value, and it apply task duplication to minimize communication overhead. The new algorithm could overcome limitations of list scheduling and task duplication algorithms.

The remainder of this paper organized as follows. Section 2 discusses problem definition. Section 3 gives an overview of the related work. Section 4 presents our developed CLDD algorithm with examples. Section 5 discusses the results and in section 6, conclusions are given.

2. PROBLEM DEFINITION
In this section, we define the models of HeDCS, a target computing system, the model of the application, and the scheduling objective [16].

A HeDCS is a set P of p processors connected in a fully connected topology. It is assumed that:

- Any p can execute a task and communicate with other at the same time because of overlapping computation and communication time.
• Once a task has started a task, it continues without interruption. Then, after completing the execution, it immediately sends the corresponding output data to all of its children in parallel.

An application is represented by a weighted Directed Acyclic Graph (DAG) $G(V,E,W)$ where $V$ is the set of nodes; and each node $v_i \in V$ represents an application task, which is an indivisible code segment that must be executed sequentially on the same processor.

$W$ is a $v \times p$ computation cost matrix in which each $w_{ij}$ gives the estimated time to execute task $v_i$ on processor $p_j$.

$E$ is the set of communication edges. The directed edge $e_{i,k}$ connects nodes $v_i$ and $v_k$, where node $v_i$ is called the parent node and node $v_k$ is called the child node. This implies that $v_k$ cannot start until $v_i$ finishes and sends its data to $v_k$. The task with no parents is called root and the task with no children is called leaf.

Fig. 1 shows an application with ten tasks. The application is represented as a DAG and the execution costs estimated for the ten tasks on the HedCS are shown as a computation cost matrix. Communication costs are written on edges of the DAG.

$$EST(t_i,p_j) = \max(T_{\text{Available}}(P_j), \max(AFT(t_k)+c_{ki})-w_{ij}) \quad (1)$$

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 $k = 1,2, \ldots , n$) on the processor $P_j$. $c_{ki}$ is the communication cost from task $t_k$ to task $t_i$, where $c_{ki}$ is equal if the predecessor task $t_k$ is assigned to processor $P_j$. For the entry task, $EST(t_{entry},P_j) = 0$.

$$EFT(t_v,P_j) = EST(t_v,P_j) + w_{ij} \quad (2)$$

Where $w_{ij}$ is the computational cost of $t_j$ on a processor $P_i$.

Performance ratio = 1 - Improvement ratio.

### Expected Completion Time

3. **EXPECTED COMPLETION TIME BASED SCHEDULING ALGORITHM**

ECTS algorithm consists of two phases namely, task prioritization phase and processor selection phase [17]. The task-prioritizing phase consists of two stages such as level wise task priority stage and task selection stage. In the first stage, the algorithm computes the priority for every task at each level by using Expected Completion Time (ECT) value. Average Computation Cost (ACC) and Maximum Data Arrival Cost (MDAC) compute the ECT value. Next Equations (3, 4, and 5) explains ACC, MDAC and ECT respectively.

$$ACC(t_i) = \frac{\sum_{j=1}^{m} w_{ij}}{m} \quad (3)$$

Where $w_{ij}$ is the estimated execution time of task $t_i$ on processor $p_j$ and $m$ is the number of processors.

$$MDCA(t_i) = \max_{k \in \text{pred}(t_i)} C_{k,i} \quad (4)$$

Where $t_i$ is the set of predecessors of task $t_i$.

$$ECT(t_i) = ACC(t_i) + MDCA(t_i) \quad (5)$$

The second stage related to the task selection in which the tasks are selected from all levels based on their priority. Moreover, in the second phase, the selected tasks are assigned to the proper processor, which minimizes its $EFT$ by using the insertion-based scheduling policy [11].

### New Scheduling Algorithm

The developed Communication Leveled DAG with Duplication (CLDD) algorithm consists of three phases, level sorting, task prioritization, and processor selection. The detailed explanation of each phase of the algorithm is given below:

#### Level Sorting Phase

Like [16], the given DAG is traversed in a top-down fashion to sort tasks at each level in order to group the tasks that are independent of each other. As a result, tasks in the same level can be executed at the same time.

#### Task Prioritizing Phase

In this phase, the CLDD algorithm selects a level and gives a priority to its tasks. It computes the priority for each task according to new attribute called Rank using Equation 6.

$$Rank(t_i) = w_{\min}(t_i) + MCC(t_i) + Max(C_{k,\text{pred}t_i}) \quad (6)$$

Where $w_{\min}(t_i)$ is the minimum weight of $t_i$, $MCC(t_i)$ refers to Mean Communication of Children (Equation 7).

$$MCC(t_i) = \sum_{m=1}^{n} C_{lm}/n \quad (7)$$

Where $n$ is the number of children, $C_{lm}$ is the communication cost between $t_i$ and its child task $t_m$. $C_{k,\text{pred}t_i}$ refers to communication between task $t_k$ (parent of $t_i$) and $t_i$. Fig. 3 shows CLDD algorithm steps. After the algorithm assigns a priority for each task in selected level, it creates a new Tasks List TL (in which the CLDD algorithm sorts all level tasks in decreasing order to execute the next phase).

#### Processor Selection Phase

This phase consists of two stages, processor selection stage, and duplication stage. Schedule Task Function in Fig. 2 is used to execute this phase.

#### Processor Selection Stage

The CLDD 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.

#### Duplication Stage

Schedule Task function also executes stage of duplication test. Attest’s, if Data Ready Time (DRT) of task $t_i$ is more than the $EFT$ of maximum parent of task $t_i$ on the same processor $p_j$, the algorithm duplicates the maximum...
parent on \( p_t \) and updates \( EFT \) of task \( t_1 \), without violating precedence constraints. Data Ready Time DRT is the idle time waited by \( t_1 \) on \( p_t \). Maximum parent is the task parent with maximum earliest finish time on different processor \( P_m \) and communication weight between this task and the maximum parent. A case study is taken into account as following.

\[
\text{Schedule\_Task}(t_i) \quad \{
\quad \text{For each (processor } P_j \text{ in the processor set } (P_j \in Q) )
\quad \{
\quad \quad \text{Compute } EFT(t_i, P_j) \text{ value}
\quad \}
\quad \text{Assign task } t_i \text{ to the processor } P_j \text{ that minimizes } EFT
\quad \}
\quad \text{If (Data Ready Time(DRT))of task } t_i \text{ on processor } P_j > EFT \text{ of maximum parent on processor } P_j
\quad \{
\quad \quad \text{Duplicate maximum parent on processor } P_j
\quad \quad \text{Update } EFT \text{ of task } t_i \text{ on processor } P_j
\quad \}
\}
\]

Fig. 2: Schedule\_Task Function.

\[
\text{Generate the DAG}
\quad \text{Sort the DAG levels according to dependency ordering}
\quad \text{For each level } L_j
\quad \{
\quad \quad \text{For each task } t_i \text{ in } L_j
\quad \quad \{
\quad \quad \quad \text{Compute } \text{Rank}(t_i) = w_{max}(t_i) + MCC(t_i) + \text{Max}(C_{\text{prop}}, b)
\quad \quad \}
\quad \quad \text{Create new Tasks List } TL
\quad \quad \text{Sort all tasks in decreasing order of Rank value in } TL
\quad \quad \text{For each task } t_i \text{ in } TL
\quad \quad \{
\quad \quad \quad \text{Call Schedule\_Task}(t_i) \text{ function}
\quad \quad \}
\quad \}
\]

Fig. 3: The Steps of the CLDD Algorithm.

Case Study:

Let us consider the application DAG shown in Fig.1 and the computation matrix. Table 1 shows stepwise trace of calculating task priority according to Equation 6. The generated schedule along with stepwise trace of the CLDD algorithm and ECTS algorithm are shown in Fig. 4.

<table>
<thead>
<tr>
<th>( L_i )</th>
<th>( T )</th>
<th>( W_{max} )</th>
<th>( \text{Max}(C_{\text{prop}}) )</th>
<th>( MCC )</th>
<th>( \text{Rank} )</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( T_1 )</td>
<td>21</td>
<td>0</td>
<td>21.4</td>
<td>42.4</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>( T_2 )</td>
<td>18</td>
<td>17</td>
<td>16.5</td>
<td>51.5</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>( T_3 )</td>
<td>27</td>
<td>31</td>
<td>16</td>
<td>74</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>( T_4 )</td>
<td>4</td>
<td>29</td>
<td>9</td>
<td>42</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>( T_5 )</td>
<td>27</td>
<td>13</td>
<td>57</td>
<td>97</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( T_6 )</td>
<td>17</td>
<td>17</td>
<td>5</td>
<td>39</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>( T_7 )</td>
<td>14</td>
<td>16</td>
<td>9</td>
<td>39</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>( T_8 )</td>
<td>23</td>
<td>11</td>
<td>42</td>
<td>76</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( T_9 )</td>
<td>8</td>
<td>57</td>
<td>7</td>
<td>72</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>( T_{10} )</td>
<td>13</td>
<td>42</td>
<td>0</td>
<td>55</td>
<td>1</td>
</tr>
</tbody>
</table>

The algorithm starts from \( T_{sort} \), to compute the rank value \( \text{Rank}(t_i) = 21 + (17+31+29+13+17)/5+0 = 42.4 \), \( \text{Rank}(t_i) = 18 + (30+3)/2+17 = 51.5 \), and so on. The schedule generated by CLDD algorithm has length of 125, while the schedule length generated by ECTS algorithm is 134. Therefore, the CLDD algorithm has shorter execution length than that the ECTS algorithm. The CLDD algorithm applies task duplication to decrease the communication overhead using idle time. When the algorithm duplicates a task, it decreases DRT of its children and decreases \( EFT \). This leads to good utilization of processors in the system.

<table>
<thead>
<tr>
<th>(a) CLDD</th>
<th>(b) ECTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig. 4: The Schedules Generated By (a) CLDD Algorithm and (b) ECTS Algorithm.</td>
<td></td>
</tr>
</tbody>
</table>

5. RESULTS AND DISCUSSIONS

5.1 Simulation Environment

To evaluate the performance of our developed CLDD algorithm, a simulator had been built using visual C# .NET 4.0 on machine with configuration: 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 CLDD algorithm with the other algorithms a set of Standard Task Graphs STG (as a benchmark) are generated varying a set of parameters that determines the characteristics of the generated DAGs [18]. These parameters are described as follows:

- **DAG size**: \( n \) (i.e. the number of tasks in the DAG).
- **Density**: We use "sameprob" and "layprob" methods to generate the DAG [19, 20].
- **Sameprob**: Let \( A \) denote a task connection matrix with elements \( a(i,j) \), where \( 0 \leq i \leq n \), and \( 0 \leq j \leq n \), represent the task number (\( t_i \) is the entry dummy node and \( t_j \) 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:
  \[
P(a(i,j)=1) = p \quad \text{for } 1 \leq i < j \leq n \quad \text{and} \quad P(a(i,j)=0) = 1-p \quad \text{for } 1 \leq i < j \leq n\]
where \( p \) indicates the probability that there exists an edge (precedence constraint) between \( t_i \) and \( t_j \).
- **Layerprob**: In this method, firstly the number of levels \( L \) in the task graph is generated. Next, the number of independent tasks in each level is randomly decided. Finally, edges between levels are connected with the same probability \( p \), as is "sameprob".
• 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 generated varying from 10, 20, 40, 60, 80, and 100 tasks. In each experiment, the probability $p$ and number of levels are assigned from the corresponding sets given below:
  - $SET_p = \{0.5, 0.6, 0.7, 0.8, 0.9\}$
  - $L = \{\text{No. Tasks}/3, \text{No. Tasks}/4, \text{No. Tasks}/5, \text{No. Tasks}/6, \text{No. Tasks}/8\}$ according to number of tasks.

5.2 Comparison Metrics and Results

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

5.2.1 Schedule Length

Schedule length is the maximum finish time of the exit task in the scheduled DAG. From Fig. 5, 6, 7, 8, 9, it is noted that the schedule length decreases after applying CLDD algorithm. Because the CLDD algorithm uses the duplication algorithm to minimize the communication overhead, the time required for finishing application execution by CLDD algorithm is better than the other algorithms. The performance ratio in schedule length is 17.6%.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{schedule_length_with_4_processors.png}
\caption{Schedule Length with 4 Processors.}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{schedule_length_with_8_processors.png}
\caption{Schedule Length with 8 Processors.}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{schedule_length_with_16_processors.png}
\caption{Schedule Length with 16 Processors.}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{schedule_length_with_32_processors.png}
\caption{Schedule Length with 32 Processors.}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{schedule_length_with_64_processors.png}
\caption{Schedule Length with 64 Processors.}
\end{figure}

5.2.2 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.

\[
\text{Speedup} = \frac{\text{Maximum Time of All Tasks on fastest processor}}{SL}
\]

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. The results of the comparative study according to the speedup parameter have been presented in Fig. 10, 11, 12, 13, 14, and 15. According to the results, it is clear that speedup of CLDD algorithm is better than speedup of the other algorithms, because all processors have finished tasks execution earlier than other algorithm so, our proposed algorithm outperforms the the other algorithms in speedup parameter. The performance ratio in speedup is 16.9%.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{speedup_with_10_tasks.png}
\caption{Speedup with 10 Tasks.}
\end{figure}
Efficiency is the speedup divided by the number of processors used.

\[
\text{Efficiency} = \frac{\text{Speedup}}{\text{number of processors used}} \quad (9)
\]

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. Therefore, efficiency of the CLDD algorithm outperforms efficiency of the other algorithms. Fig. 16, 17, 18, 19, 20, and 21 show the efficiency of the CLDD algorithm compared with HEFT, CPOP and ECTS algorithms. The performance ratio in efficiency that has been achieved by CLDD algorithm is 16.6%.
5.2.4 Time complexity

Time complexity is the amount of time taken to assign every task to specific processor according to specific priority. The CLDD algorithm used depth first search to sort the DAG levels with time complexity O(n+e) where n is the number of nodes, and e is the number of edges. And it used the binary heap to implement the priority, which has a time complexity of O(log n). So the time complexity of the priority phase is O(n log n). The time complexity of processor selection is O(np). Finally total time complexity equal O(np+n log n+n+e). Time complexity of CLDD algorithm may be approximated to O(np).

Insertion based technique increases time complexity, because the algorithm search in all processors for processor with proper idle time to assign specific take into it. In CLDD algorithm, we replace this technique by task duplication with low time complexity. From Table 2, we observed that, CLDD algorithm has the lowest time complexity.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Reference</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>HCNF</td>
<td>[15]</td>
<td>O(n^2 log n)</td>
</tr>
<tr>
<td>ECTS</td>
<td>[17]</td>
<td>O(n^2p)</td>
</tr>
<tr>
<td>CLDD</td>
<td>OURS</td>
<td>O(np)</td>
</tr>
</tbody>
</table>

6. CONCLUSIONS

In this paper, a new Communication Leveled DAG with Duplication CLDD algorithm is presented for heterogeneous distributed computing systems (HeDCS). This algorithm is based on rank value to give a priority to each task. The CLDD algorithm also uses task duplication with low time complexity to minimize communication overhead. According to the simulation results, it is noted that the CLDD algorithm is better than ECTS, HEFT, CPOP algorithms in terms of time complexity, schedule length, speedup and efficiency. Performance ratio in schedule length, speedup and efficiency respectively are 17.6%, 16.9% and 16.6%. The CLDD algorithm can be tested on real applications and the development can be made on efficiency. The algorithm can be applied on cloud and grid systems as a future work.

7. REFERENCES


[18] URL: http://www.kasahara.elec.waseda.ac.jp/schedule/making_e.html
