CHAPTER 3

Duplication with Task Assignment in Mesh Topology

Distributed System consists of numerous self-ruling processors that communicate via interconnection network. As previous chapter discussed that, each network follows different connectivity architectures, known as network topology. Mesh topology is one of the topologies [L. N. Bhuyan, 1984] that are employed in this chapter for network connectivity. However, handling of mesh topology is very difficult because of the inter-connectivity between every node. Such network connectivity among processors can be of homogeneous or heterogeneous type. Homogeneous systems share identical architecture, whereas diverse architecture flows in heterogeneous. Therefore, task scheduling is complicated in heterogeneous systems due to non-uniform speed and communication bandwidth. List-based and cluster based are two important scheduling classes that help in task scheduling of heterogeneous systems [D Bozdağ, 2006]. In order to resolve the complication of processor heterogeneity, the author has used cluster based scheduling. Based on processor computational capacity [Y. Jegou, 1997] entire system splits into three clusters (High, Medium and Low).

Parallel task execution is the primary advantage of distributed system. In parallel execution independent subtasks executes correspondingly on various processors. These subtasks are generated from single task that is called as a Directed Acyclic Graph (DAG) that shows interdependency in-between subtasks. In order to accomplish the complete DAG as fast as possible, subtasks are allocated to separate processors on the given system. These processors execute allocated tasks in parallel according to their computational speed. After achieving the results, destination processor transmits it to the source processor (origin) of tasks. This chapter explains the strategic duplication of tasks on the various processors that finally reduces the schedule length of the entire DAG.

Execution of any task passes through following two heuristics:
1. Partitioning heuristic under which tasks split into dependent/independent tasks known as DAG [J. Lopez, 2000]. This DAG represents the size of each task along with computational power consumption.

2. Allotment of processors to these distributed sub-tasks is another phase. First-Fit, Worst-Fit, Best-Fit and Communication aware worst-fit (CAWF) are some task assignment heuristics [J. Lopez, 2000; A. Burchard, 1995 & C. Wang, 2007] that works with/without task duplication.

These partitioning and assignment heuristics come under the scheduling problem. This problem is also known as grain size determination [J.E.G. Coffman, 1996], clustering problem [A. Bashir, 2013 & J. Baxter, 1989] or internalization pre-pass [B. Kruatrachue, 1988].

First-Fit, Worst-Fit and Best-Fit heuristics work in a sequential manner and duplication of task is not followed here. CAWF is designed for the reduction of communication cost in which two dependent tasks (predecessor-successor) can be allocated on a same processor that reduces the communication cost between tasks. In case of multiple successors of single predecessor, CAWF assigns one of the successors on the same processor with its predecessor and rest successors use worst case heuristic for allocation. Hence, this is the downside of CAWF algorithm.

This chapter discusses a new task duplication method that will overcome the limitation of CAWF. Author has chosen basic heuristic algorithms (where duplication is not allowed), CAWF and HEFT-TD algorithms to compare with proposed algorithm. Since these algorithms have their own properties, time complexities and advantages during task assignment. There are many other algorithms for the execution of DAG in heterogeneous environment, i.e. DBUS and HEFT-TD [D. Bozdag, 2006 & P. Claudel, 2010] algorithms (few properties are comparable to proposed algorithm with different approach).

In this division, author has projected a duplication of task at the time of its allocation before the execution. In this projected algorithm, DAG is traversed from bottom to up approach that checks the interdependencies of tasks. If two independent tasks are found then those tasks will execute independently (parallel). Next section will discuss about some existing scheduling algorithms of task duplication along with proposed task duplication algorithm followed by its performance.
3.1 Existing Scheduling Algorithms

Load balancing is one of the main approaches of distributed system. This load balancing is accomplished by using task duplication or migration in-between processors. As we are dealing with dependent tasks, duplication of tasks is employed here. Main role of task duplication is to reduce the communication cost that helps in diminution of the overall schedule length of entire DAG. Many researchers have suggested various strategies of task duplication [S. Ranaweera, 2000; P. Chaudhuri, 2010 & J. Singh, 2012]

DAG is an arrangement of multiple tasks, out of which some tasks are dependent on previous tasks and some are independent. In case of dependency, successor tasks could not execute before the execution of dependent predecessor tasks. On the other side, independent tasks can execute in parallel on several processors. In a DAG, \( G = (V, E) \), \( E \) is a link between two nodes that explains the communication cost between two dependent tasks. These subtasks (tasks) are assigned to various processors based on following features already discussed in many other papers [P. Chaudhuri, 2010; R. Sharma, 2011 & S. Ranaweera, 2000]:

Definition 3.1: Computation cost (Execution time) of any task on a given processor is dependent on the computational capacity of a particular processor. Time taken by a processor to execute a particular task is known as the computation cost. Computation cost also depends on the size of task as well.

Consider \( CC(t_i,k) \) is the computation cost of task \( t_i \) on \( k^{th} \) processor from \( p \) number of processors. Hence, the average computation cost of any task \( CC(t_i) \) is defined as:

\[
CC(t_i) = \frac{\sum_{k=1}^{p} CC(t_{i,k})}{p}
\]  
\((3.1)\)

Definition 3.2: Communication Cost \( C_{t_i,t_j} \) is the time consumed by the processor in sending data (results) of one task to another processor. This communication cost is dependent on the volume of communicating data and data transfer rate from source to destination processor [P. Chaudhuri, 2010 & J. Singh, 2012].

\[
C_{t_i,t_j} = S_v + \frac{v(t_i,t_j)}{D_{x,y}}
\]  
\((3.2)\)
If two jobs are assigned on same processor, then the communication cost, \( C_{t_i,t_j} = 0 \).

*Definition 3.3:* Total Finish Time (TFT) [E.G. Coffman, 1998]: Total finish time of \( k^{th} \) tasks on \( P_n \) processor is

\[
TFT(P_n, k) = \sum_{i=1}^{k} (\text{new arrival}(T_i) + \text{Execution time}(T_i))
\]

where

\[
\text{new arrival}(T_i) = \text{Execution time}(\text{pred}(T_i)) + C_{\text{pred}(T_i), T_i}
\]

\( P_n \) is the number of processors i.e. \( P_1, P_2, P_3, \ldots \ldots , P_n \) and \( k \) are number of tasks scheduled on given processor. Hence, \( TFT(P_n, k) \) is total finish time of \( k^{th} \) task on \( n^{th} \) processor.

![Fig. 3.1. Arbitrary DAG with Communication Cost](image)

Figure (3.1) explains the DAG that contains tasks (subtasks) \( \{T_1, T_2, T_3, T_4, T_5, T_6\} \) and \( \{25,30,50,65,70,15,25\} \) are their respective communication costs in-between the dependent tasks. Later on, the generation of subtasks (tasks of the DAG) will be assigned to respective processors. Task assignment is the process of multiple task allocation to the numerous processors along-with parallel allocation and execution method for the same [Lo. V.M., 1988].

In distributed system, the selection of processors for task allocation can be sequential or parallel. For sequential task allocation First Fit (FF), Best Fit (BF) and Worst Fit (WF) are well known. All these mentioned sequential allocation heuristics focuses on computation costs but not on communication costs. In [C. Wang, 2007] author has discussed an assignment heuristic approach that focuses on communication cost along with computation cost. This heuristic is known as communication aware worst fit (CAWF). According to CAWF, same processor is
assigned to a pair of predecessor-successor that brings down the communication cost in-between assigned pair. However, if one predecessor has multiple successors then the worst fit algorithm is used for rest successors. Although, sequential assignment of tasks is also present here but this algorithm seems helpful in reducing the communication cost.

Equation (3.3) calculates the total finish time of complete DAG on a particular processor. This TFT is dependent on the execution cost of every subtask (task) on the respective processors and communication cost between dependent tasks (subtasks). Table (3.1) explains the execution cost (computation cost) of tasks on respective processors:

<table>
<thead>
<tr>
<th>$T_i$</th>
<th>$P_j$</th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$P_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>35</td>
<td>5</td>
<td>15</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>$T_2$</td>
<td>9</td>
<td>4</td>
<td>10</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>$T_3$</td>
<td>6</td>
<td>8</td>
<td>4</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>$T_4$</td>
<td>23</td>
<td>45</td>
<td>15</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>$T_5$</td>
<td>10</td>
<td>7</td>
<td>9</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>$T_6$</td>
<td>30</td>
<td>9</td>
<td>5</td>
<td>18</td>
<td></td>
</tr>
</tbody>
</table>

This section is working on distributed system, parallel execution and allocations of tasks are considered here. Let us consider the case when processors are selected randomly for task assignment and execution as well. In figure (3.2), $P_3$ processor is selected randomly for $T_4, T_4$ and $T_6$ tasks; $P_2$ is assigned to $T_3$ and $T_5$; similarly $P_1$ execute $T_2$ task. Based on their execution cost on assigned processors and communication costs between tasks overall DAG schedule length has been calculated. In arbitrary selection, DAG schedule length may vary because it is dependent on preferred processor. There is no criterion of processor selection for task execution in arbitrary method.
Fig. 3.2. Arbitrary allocation of tasks on processors of distributed system

\[ DAG \textit{Schedule Length} = \max_{1 \leq n} TFT(P_n) \quad (3.5) \]

\[ = \max(TFT(P_1), TFT(P_2), TFT(P_3)) = \max(39, 96, 138) \]

\[ = 138 \text{ unit of time} \quad (3.6) \]

Now, in figure (3.3) tasks are assigned according to CAWF algorithm. Tasks having predecessor and successors are allocated to the same processor and another task will follow worst fit. In figure (3.1) \( I_1 \) is the only predecessor of tasks \( I_2 \) & \( I_3 \). Similarly, \( I_2 \) is predecessor of tasks \( T_4 \) & \( T_5 \). According to CAWF, one of the successors of these predecessors will allocate on the same CPU and other tasks following worst fit. Therefore, \( T_1 \) and \( T_3 \) (dependent tasks) are assigned on processor \( P_1 \). Similarly, \( T_2, T_4, T_5 \) and \( T_6 \) are interdependent tasks and sequentially assigned to next processor \( P_2 \). Lastly, on the basis of computation and communication cost DAG schedule length has been calculated which is lesser than the previous method due to reduction in communication costs.

Fig. 3.3. DAG Execution using CAWF heuristics

\[ DAG \textit{Schedule Length} = \max(TFT(P_1), TFT(P_2)) = \max(41, 165) \]
Further, third type of allocation is proposed task duplication algorithm that is essentially an advanced adaptation of CAWF. In this method, tasks having less execution cost as compared to communication cost becomes a duplicated task on a given processor.

From above example task $T_1$ duplicates on $P_4$ processor because its communication costs towards dependent tasks $T_2$ and $T_3$ is greater than its computation cost on particular processors. Similarly, the computation costs of other dependent tasks are greater than their communication costs and therefore those tasks will not duplicate on other processors. By applying such duplication technique, overall schedule length of DAG is comparatively lower than the previous methods.

![Diagram showing task duplication and communication cost comparison]

**Fig. 3.4. Proposed Task Duplication methodology**

\[
\text{DAG Schedule Length} = \max(TFT(P_2), TFT(P_3), TFT(P_4)) = \max(13, 103, 28)
\]

\[= 103 \text{ units of time} \quad (3.8)\]

The proposed duplication algorithm is somewhat similar with HEFT-TD and DRUS algorithms. Additionally, the approach used here is different. The approach used in HEFT-TD in top-down whereas bottom-up approach is used in present scenario. Therefore, proposed algorithm gives alike or little bit improved result than the existing. Next section will explain the new algorithm of task duplication followed by simulation results.

### 3.2 Task Duplication Assisted Schedule Length Minimization (TDASLM) Algorithm
There are many approaches has been used for task assignment \textit{i.e.} First-fit, Best-first, Worst-fit and CAWF etc. but all these heuristics, select processors sequentially (first processor assigns first and so forth) for the assignment of tasks without duplication. CAWF algorithm reduces the communication cost by assigning the predecessor and successor on a single processor. This approach works fine if the predecessor has single successor therefore, downside of CAWF approach is multiple successors of single predecessor. As we recognize that, the primary motive of task duplication is to reduce the communication cost that affects the overall schedule length of DAG. Hence, in order to overcome the problem of CAWF task duplication methodology has been used. Now a day’s numerous researchers have designed many task duplication algorithms [D.Bozdag, 2006; P.Chaudhuri, 2010 & J. Singh, 2012] with different approaches.

Topology we are using is a mesh that connects every processor with each other processor of the system. After the generation of DAG on given processor, proposed algorithm uses Bottom-up traversing of DAG similar to DBUS algorithm [D.Bozdag, 2006]. This approach determines the dependency and independency in-between sub-tasks of DAG. Independent tasks can execute in parallel and duplication is used for dependent tasks. Task assignments depend on the computational capacity of an assigned processor because the job will execute on allotted processors. Duplication of task is based on the communication cost and execution cost of processors. At the time of duplication, few critical things must remember:

1) Limited number of duplicates: Algorithm must understand the number of duplications of any task (Successor/Predecessor). Avoid useless duplication of tasks. Consider the $C_{r_i,t_j}$ between $i^{th}$ and $j^{th}$ task is less than the $C(P_j)$ of $j^{th}$ processor then there is no need of task duplication.

2) By Bottom-up traversing of DAG all child tasks execute first and then parent task. Due to which parent task duplication decreases.

In the remainder of this section different module of task duplication has elucidated.

\textbf{3.2.1 Clustering of Heterogeneous Processors with Mesh Topology}

Mesh topology is used here for the interconnection of heterogeneous processors. Therefore, processor computational power shows a discrepancy. In order to handle this heterogeneous
behavior of system, the complete distributed system splits into three clusters (based on computational capacity) i.e. High, Medium and Low. For the grouping of processors, we have fixed some range that determines the efficiency level of processors. These ranges make a decision randomly from 0 to 10.

Fig. 3.5. Clustering of Processors

In figure (3.5), each cell represents a node (processor) and based on efficiency range complete system divides into three groups:

Blue color represents “Low efficiency” that comes under 0 to 4 ranges. Yellow color represents “Medium efficiency” and this range lies between 5 and 7. Lastly, Red color is for “High efficiency” and its range lies from 8 to 10.

Along with efficiencies these nodes also have communication costs in-between and author represent this cost with the help of adjacency matrix. Figure (3.6) is a matrix of communication costs between several CPUs. For example: $C_{3,2}$.

Fig. 3.6. Communication costs between nodes
3.2.2 Generation of Task on Nodes

In a distributed heterogeneous system, DAG’s (tasks) can be generated on any node at any time. In figure (3.7), task generation on particular processor is indicated by green color.

![Task Generation on any node of system](image)

Fig. 3.7. Task Generation on any node of system

Following is the task generation algorithm:

The algorithm generates task randomly on any node and by getEfficiency() function retrieve the efficiency of a particular node.

BEGIN

\textbf{TASK\_EXECUTION\_ACTION\_PERFORMED (java.awt.event.ActionEvent evt)}

1. Calendar c= Calendar.getInstance()
2. long m=c.getTimeInMillis()
3. Random r=new Random(m)
4. xcor=r.nextInt()
5. m=c.getTimeInMillis()
6. r.setSeed(m)
7. ycor=r.nextInt()
8. jArray[Math.abs(xcor%5)][Math.abs(ycor%5)].setBackground(Color.GREEN)
9. group.getEfficiency(Math.abs(xcor%5),Math.abs(ycor%5))

END

After the generation of DAG following algorithm retrieve the efficiency of that node and its communication cost with near (other) nodes.

getmatrix() function obtain the communication costs from one processor (where task generate) to other nodes. gettaskmatrix() function set the DAG on particular node.
SHOW-ACTION-PERFORMED (java.awt.event.ActionEvent evt)

BEGIN
  1. ndag.getmatrix()
  2. tdag.gettaskmatrix()

END

3.2.3 **DAG Matrix and its Tracing**

Above module is the basic framework of simulation. This module explains the DAG (in matrix format) of tasks that shows the dependency/independence between tasks. In DAG matrix, 0 represents an independent task and 1 represents a dependency between two tasks (figure (3.8)).

![DAG Matrix Image]

**Fig. 3.8.** DAG representation in terms of matrix

For the execution of complete DAG Bottom-up approach is used. Task $T_7$ is independent task (Column of $T_7$ contains 0), $T_6$ is dependent on $T_7$ ($T_6$ column has 1 on $T_7$ row). Similarly, other dependencies have been made. For traversing of this matrix of tasks (DAG), first, we check the dependencies (occurrence of 1's in a column) and based on this occurrence sorting of tasks are done. This computation takes $O(n^2)$ time.

---

**Input:** A sequence of $n$ subtasks of DAG ($t_1, t_2, t_3, \ldots, t_{\text{taskDAG}}$).

**Output:** DAG in terms of matrix has been generated.

BEGIN

<table>
<thead>
<tr>
<th>Count</th>
<th>Cost</th>
<th>Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>count=0</td>
<td>$c_1$</td>
<td>1</td>
</tr>
<tr>
<td>for $i=0$ to taskDAG.length</td>
<td>$c_2$</td>
<td>$n + 1$</td>
</tr>
<tr>
<td>for $j=0$ to taskDAG.length</td>
<td>$c_3$</td>
<td>$n^2$</td>
</tr>
<tr>
<td>if taskDAG[$j$][$i$]==1</td>
<td>$c_4$</td>
<td>$n$</td>
</tr>
<tr>
<td>count++</td>
<td>$c_5$</td>
<td>$n$</td>
</tr>
</tbody>
</table>

6. End for

7. End for

END
Hence, in the worst case, the running time of DAG generation is

\[
T(n) = c_1 \cdot 1 + c_2 \cdot (n+1) + c_3 \cdot n^2 + c_4 \cdot (n) + c_5 \cdot (n) \\
= c_1 + c_2 \cdot n + c_2 + c_3 \cdot n^2 + c_4 \cdot n + c_5 \cdot n \\
= c_3 \cdot n^2 + (c_2 + c_4 + c_5)n + (c_1 + c_2) - O(n^2).
\]

Running time of algorithm is the sum of running times for each executed statement. Above equation can be expressed in the form of \(an^2 + bn + c\) for constants \(a, b,\) and \(c\) that again depends on statement costs \(c_i;\) it is thus a quadratic function of \(n\ i.e.\ n^2.\)

After getting the dependent tasks, scheduler checks whether this dependency is direct or indirect. For example, in figure (3.8) task \(T_6\) directly dependent on \(T_7\) task and \(T_2\) is indirectly dependent on \(T_6\ (T_2 \rightarrow T_4 \rightarrow T_6). These dependency are determined by using Boolean matrix multiplication.

**Input:** Two copies of generating DAG of tasks for Boolean Matrix Multiplication.

**Output:** Dependency of tasks.

**CHECK-INDIRECT-DEPENDENCY (matrixsize1[][],matrixsize2[][],Row, Column)**

BEGIN

1. \(m = ((\text{matrixsize1.length}) \times (\text{matrixsize1.length}))/2\)
2. for \(i=\text{Row to matrixsize1.length}\)
3. \(\text{ResultMatrix} = \text{new int}[(\text{matrixsize1.length})][\text{matrixsize1.length}]\)
4. for \(j=\text{Column to matrixsize2.length}\)
5. \(\text{rowVector} = \text{getCurrentRow(matrixsize1, i)}\)
6. \(\text{columnVector} = \text{getCurrentColumn(matrixsize2, j)}\)
7. for \(k=0 \text{ to matrixsize2.length}\)
8. \(\text{if rowVector}[k] == 1 \&\& \text{columnVector}[k] == 1\)
9. \(\text{ResultMatrix}[i][j] = 1\)
10. \(\text{flag} = \text{true}\)
11. \(\text{break}\)
12. \(\text{End if}\)
13. \(\text{End for}\)
14. \(\text{Flag} = \text{false}\)
15. \(\text{if !flag}\)
16. \(\text{ResultMatrix}[i][j] = 0\)
17. \(\text{End for}\)
18. \(\text{End for}\)
19. \(\text{for} i = \text{Row to matrixsize2.length}\)
20. \(\text{for} j = \text{Column to matrixsize2.length}\)
21. \(\text{End for}\)
22. \(\text{End for}\)
23. \(\text{if ResultMatrix[Row][Column] == 1}\)
24.    return true
25.  else
26.    matrixsize1 = ResultMatrix
27.  End for
28.  return false
End CHECK-INDIRECT-DEPENDENCY()

Similar to above algorithms running time, author examine that all rows of giving matrix have \(\log n\) elements, each of which is either 0 or 1. Similar examination happens with columns of the given matrix. In Boolean matrix multiplication, divide complete matrix into rows and columns and each row (column) is having \(\log n\) elements. Therefore, here the complexity is \(O\left(\frac{1}{\log n}\right)\). The first for loop calculates the number of multiplications (number of intermediate nodes from one task to another) and within it Boolean multiplication between matrices having \(O(n^2)\) complexity. Hence, the overall running time here is \(O(n^3 / \log n)\).

This traversing of DAG gives a set of dependent and independent tasks. Further, this set adjoins the Queue of sets that works as a dispatcher. The purpose of a dispatcher is to discharge the tasks on the nodes; Task sets come in front executes in parallel on different processors and next set is dependent on that previous set. This operation dispatch sets one by one, so, it is taking \(O(1)\) time.

\textbf{Input:} Independent or dependent tasks add into a queue.

\textbf{Output:} Dispatch tasks for execution.

\begin{verbatim}
QUEUE<SET<STRING>> QUEUEOFSET()

BEGIN

1. Set<String> s = IndependentTaskset()
2. if (setqueue.isEmpty())
3.   setqueue.add(s)
4. return setqueue

End QUEUEOFSET()

QUEUE<SET<STRING>> TASKEXECUTION()

1. Queue<Set<String>> q = queueofset()
2. while (q.iterator().hasNext())
3.   Taskexecution(q.element())
4. return setqueue

End TASKEXECUTION()

\end{verbatim}
Above function `QUITUEOFSET()` add the returned set of independent task and other function dispatches the sets for execution. Tracing and dispatching of the tasks of DAG takes $O(n^3/\log n)$ time in total.

### 3.2.4 Assignment without Duplication

The previous module is actual backbone of complete simulation. Dispatcher dispatches the independent tasks on the nodes and execution of the project will continue on assigned processor.

Figure (3.5) shows the clusters of processors and Table (3.1) represents the computation cost of processors with respect to tasks. Execution of tasks from the dispatcher depends on the priorities of tasks. Here, queue for a set of tasks has maintained that follows FIFO criterion.

![Dispatcher Queue (FIFO)](image)

The above dispatcher works on every processor separately. $T_5, T_7$ tasks will execute parallel on different processors. Now $T_6$ is dependent on $T_7$, after getting the result from $T_7$, $T_6$ assign to other processor. $T_3, T_4$ are dependent on $T_6$. After getting output from $T_6$, $T_3$ and $T_4$ can execute in parallel. Now $T_2$ requires output from $T_5, T_3$ and $T_4$. Finally $T_2$ executes on its own processor (source).

In figure (3.7), random tasks generate on four different processors having different efficiencies. Let us take above explained DAG that generates on high efficiency processor. In arbitrary assignment heuristic algorithms, dispatcher assigns task on other processors randomly. If its neighbor node will unable to execute more tasks than a source processor will switch to another processor.

### 3.2.5 Duplication Scheduling Explanation

This module explains the proposed duplication strategy that helps in diminution of schedule
length of DAG. After the generation of DAG (task) its computational capability (efficiency) and communication cost on other processors is calculated. After bottom-up tracing of DAG dispatcher queue is maintained that initially allocate processors to the first set of independent tasks. Those assigned independent tasks can execute in parallel on allocated processors. After the execution of assigned tasks, processor of dependent tasks starts execution because output of predecessor becomes the input for its successor.

Now, for the execution of such dependents, task duplication is used. Our duplication approach is based on following factors:

1. Communication cost: Time taken in the resettlement of the predecessor output towards its successor is the communication cost between them. If this data transfer rate is high then there is a requirement of duplication.

2. Computation cost: We all are aware that the time occupied by a processor to execute the specified task is the computation cost of the assigned tasks on allotted processor.

In order to execute our approach, first we set computation costs of particular task (let us say task\(_i\)) on all processors in ascending order. Additionally, communication costs between task\(_i\) and its successors will arrange in descending order. Afterwards, scheduler compares the successors computation cost on source processor of task\(_i\) and communication cost between tasks. If computation cost is smaller than the \(C_{task_i,successors}\) then duplication of successor task on source processor of task\(_i\) is achievable. This way of duplication along with bottom-up approach decreases the number of duplications also. Following algorithm is explaining the conditional duplication of our approach.

---

**Input:** task with execution time (ET) and communication cost \((C_{t_i,t_j})\) between connected tasks.

**Output:** Duplicate tasks to the destination processor (DP).

BEGIN

1. **IF** (ET < \(C_{t_i,t_j}\))

2. **DUPLICATE** \((T_i, DP)\)

3. **ELSE**

4. setqueue. **TASKEXECUTION** \((T_i, P)\)

END
During the simulation of this duplication algorithm, author suspect that the number of processors affect the schedule length of the complete DAG with or without duplication. In it, one common DAG is simulated on two different distributed systems with or without duplication. The schedule length of DAG varies from the number of processors. Author checked it for 5 and 10 processors.

**Question 3.1:** If we increase the number of processors in any distributed system then, can there be a need of task duplication?

**Explanation:** Addition of any processor in a system means accumulation of new computational power in the same. We can say that if we are increasing the number of processing powers than schedule length of DAG should become small even without duplication.

Let us assume following common DAG and two different pairs of distributed system. One system is a group of 5 processors. Other system is a group of 10 processors.

Figure (3.10) explains the computation costs of tasks on given processors of the system. This theorem explains the relation between task duplication and schedule length. In order to establish the relation between both, let us consider following two examples:

1. **Less number of processors with or without duplication:**
   Figure (3.10(b)) is a system of 5 processors with general computational capacity. If we executes given DAG (figure (3.10 (a))) on this system by using duplication, overall schedule length of DAG is comparatively low (as shown in figure (3.11)).

2. **More number of processors with or without duplication:**
   After the implementation of small system, we expand the given system by the addition of 5 supplementary processors to extra computational capacity. Following the execution of same DAG on this new arrangement, we again figure out that the schedule length of the DAG is less by using duplication.

For task duplication, author uses the following criteria:

If $(ET < C_{t_i, t_j})$ then duplication of task occurs but if reverse happens then there is no need of duplication.
Other side of the coin is that when we increase the limit of processors by 5 then DAG schedule length has been increased as compared to 5 processor systems. Consequently, we cannot say that schedule length is dependent upon size of system. By increasing the number of processors, overall schedule length may or may not be reduced without duplication. Reason behind it, that execution of a task is dependent on computational capacity of any processor of the system and usage of duplication is best way to shorten the schedule length. Figure (3.11) is showing the result of given theorem.
3.3 Results and Comparisons

The proposed algorithm for task duplication in heterogeneous system with mesh topology is simulated. Simulation results on Bottom-up approach of random DAG shows that the makespan generated by the proposed algorithm is better than the existing arbitrary task assignment heuristics, CAWF and HEFT-TD algorithm. The concept of Task Duplication is used in Task Assignment Heuristic in Mesh Topology. This new algorithm is named as Task Duplication Assisted Schedule Length Minimization Algorithm (TDASLM). The given example and simulations performed, explain that in the given case total finish time can be cut down by reducing the communication cost because of duplication using optimal assignment (communication cost must be greater than Execution time of related tasks on that processor).

3.3.1 Experimental Set-up and Test Bed

Figure (3.12) explains the experimental setup of proposed study. Following are some attributes that explain the functioning of given set-up:

1. Topology

In a distributed system, connectivity architecture follows by the processors of the entire system is known as topology. Some basic topologies followed by any network/ distributed system are BUS, Ring, Star and connected Mesh topologies. The implementation of BUS, Ring and Star topologies are simpler than connected Mesh topology. In connected Mesh
topology, each processor is associated with every other processor of the system. Due to the connectivity complexity of mesh topology its handling is difficult to enforce. We simulate mesh topology in our proposed work.

![Experimental Set-up of Proposed Work](image)

Fig. 3.12. Experimental Set-up of Proposed Work

2. Participating processors

Participating processors are the processor that belongs to distributed system. The participation of processors devises an environment of the system that determines the overall performance of the system. Here, heterogeneous processors are utilized in this simulation. Heterogeneous means, each processor of the system share different architecture. Internal storage capacity and computational power are the main components of any architecture. In heterogeneous, every processor has different computational capacities. Hence, we have used clustering method that split the entire system into three clusters i.e. Low, Medium and High. All clusters have some fix range of computational efficiency (Figure (3.5)).

3. Normal DAG subtasks

The proposed duplication algorithm is working on DAG. As previous section discussed that independent tasks will execute in parallel on different processors. Those assigned tasks behave like normal executable tasks.
4. Duplicate subtasks

The entire DAG (task) is divided into dependent or independent tasks. Duplication methodology is used to decrease the communication cost between dependent tasks. There are various methods for task duplication but the way of processor selection for the execution of duplicate tasks/subtasks varies. This proposed algorithm compares the computation and communication cost of duplicated task on destination processor. If its computation cost on processor is greater than the communication cost than there is no requirement of duplication.

These above techniques and all components of the framework are implemented in Netbeans 6.9 IDE environment running with Ubuntu Version 11.10. Periodic generation of random DAG takes place on any processor. Matrix is used to execute the DAG and Queue data structure has been used to implement the dispatcher. Java threads are used to execute and communicate subtasks with each other. 100 DAG's up to 30 times on 12 and 16 processors run in order to compute the overall schedule length of DAG. With this new duplication algorithm CAWF, Arbitrary task assignment heuristics and HEFT-TD scheduling algorithms are simulated on above designed framework.

Proposed algorithm is the reproduction of HEFT-TD [P. Chaudhuri, 2010], but it is implemented by using mesh topology (loosely coupled distributed system) and bottom-up approach. Therefore, its complexity is high.

<table>
<thead>
<tr>
<th>Duplication Algorithms</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBUS</td>
<td>$O(</td>
</tr>
<tr>
<td>HEFT-TD</td>
<td>$O(</td>
</tr>
<tr>
<td>TDASLM(Proposed Algorithm)</td>
<td>$O(n^3/\log n)$</td>
</tr>
</tbody>
</table>

Mesh topology is good for a limited number of processors. As processors increases connectivity’s between them also increases, due to which system becomes more complex. It is the limitation of proposed algorithm that this algorithm is finer for inadequate size of distributed system.
3.3.2 Comparisons:

3.3.2.1 Schedule Length:

Schedule length (TFT) of \( \text{DAG} \) is computed by using the equation (3.3). Total finish time of \( \text{DAG} \) without duplication (Arbitrary processor selection method) is very high as compared to \( \text{CAWF} \) where schedule length is decreased by cutting down the communication cost in-between tasks. When duplication is used, the resultant schedule length is very low as compared to \( \text{CAWF} \) and arbitrary method as well. As HEFT-TD method uses top-down approach in \( \text{DAG} \) traversing in multiprocessor (tightly coupled distributed system) and proposed algorithm employs a bottom-up approach on loosely coupled distributed system. Therefore, schedule length of proposed algorithm gives alike or little bit well results than other two algorithms.

![Random Generated DAG vs Schedule Length](image)

**Fig. 3.13.** Comparison of proposed algorithm with existing assignment algorithms

3.3.2.2 Computation to communication Ratio (CCR):

Computation to communication ratio (CCR) is the ratio of number of calculations a process does to the total size of the messages it sends. This ratio depends upon the average communication volume and average task execution weight. Speed of communication channel also affects the CCR and this speed depends on the computational speed of processors. In this chapter, author has used heterogeneous processors having different computational speed. It is a mesh topology also, so high processing power processor may connect with low processing power processor and vice versa is possible. Therefore, if any data moves from the higher efficiency processor to less efficient processor and speed of communication channel is very fast than CCR will be higher but if the speed of channel is high and computational cost of
processor is very low than CCR will again get affected. Therefore, CCR is varying with both processor speeds as well as a communications channel because mesh topology with heterogeneous processors use here.

3.4 Summary

The task duplication concept has been employed here during assignment procedure (before implementation of tasks). This duplication reduces the total finish time of task. By theorem 3.1, author also explained that total finish time (schedule length) of task is utterly dependent upon the execution power of the processor and if duplication is used then it will generate good results. During simulation of task duplication author realizes that duplication of task too can overload a processor also. Hence, in order to overcome this overload problem author has used task migration methodology. How this migration technique is used with task duplication will be explained in next chapter. Therefore, further author extend this algorithm with task migration in Distributed System.