CHAPTER 3

IMPROVED RELIABLE HETEROGENOUS EARLIEST
FINISH TIME ALGORITHM

3.1 INTRODUCTION

Heterogeneous Distributed Computing Environment (HDCS) has a diverse set of high performance processors, connected with high speed networks to solve computationally intensive applications. The execution of these applications in such an environment requires effective scheduling strategies that take into account, both algorithmic and architectural characteristics to achieve a good mapping of tasks to processors, i.e., to minimize the schedule length (Khokhar et al 1993). There are several algorithms by (Sih and Lee 1993, Wang et al 1997, Kwok and Ahmad 1999, Topcuoglu et al 2002) addressing the problem of scheduling, having the objective of minimizing the length of the schedule.

List scheduling based heuristics usually generate good quality schedules at a reasonable cost. Various methods to specify the priorities of nodes and select the best processor have been proposed by (Kwok and Ahmad 1999, Topcuoglu et al 2002). List scheduling heuristics are originally designed for homogeneous systems where processors speed, network bandwidth between any processors remains the same. It has been extended in two directions. Firstly, several dynamic list scheduling algorithms have been introduced by (Sih and Lee 1993, Yang and Gerasoulis 1994). These algorithms update the priorities of each node and the scheduling list
dynamically at each step. Similar to traditional list scheduling algorithms, at each step the node with the highest priority is selected for scheduling. Dynamic list scheduling can potentially generate better schedules. However, these approaches can significantly increase the time complexity of the algorithms. Secondly, a number of list scheduling algorithms for heterogeneous environment have been proposed (Radulescu and Gemund 2000, Topcuoglu et al 2002). A comparison of those algorithms reveals that during the processor selection phase: (1) insertion-based policy, which allows the possible insertion of a task in an earliest idle time slot between two already-scheduled tasks on a processor, is better than non-insertion based counterparts. (2) Processor selection criteria that consider the Earliest finish Time outperform those do not include this factor.

Although scheduling in general is a well studied problem, most of the algorithms consider only the single objective of minimizing the schedule length. However, as heterogeneous systems become larger and larger, the issue of reliability of such an environment needs to be addressed. The probability of failure of the resource is more if load is not balanced equally on all the resources thus affecting the reliability of the schedule. Thus, there is a need for extending the list scheduling algorithms such that it minimizes the schedule length and also the failure probability (Attiya and Hamam 2004, Dogan and Ozguner 2005, Pop et al 2007, Tang et al 2010).

In this chapter, a new algorithm namely Improved Reliable Heterogeneous Earliest Finish Time (IRHEFT) algorithm is proposed by extending the traditional HEFT algorithm to consider the reliability. The impact of incorporating the reliability cost functions of RDLS and a factor called load difference with HEFT, to maximize reliability of the schedule and minimizing the schedule length are discussed. The processor workload is represented by the time the processor takes to execute all tasks assigned to it.
The load difference factor is calculated using this workload among the processors. The performance of the proposed algorithm is demonstrated by using it for randomly generated task graphs and a real application task graph.

### 3.2 SYSTEM MODEL

A scheduling system usually consists of three parts: application, computing environment, and scheduling goal. The application and computing environment can be represented by a task graph and resource graph respectively.

#### 3.2.1 Task Graph

The DAG is a generic model of a workflow application consisting of a set of tasks (nodes) among which precedence constraints exist. It is represented by $G = (V, E)$, where $V$ is the set of $n$ tasks that can be executed on a subset of the available processors. $E$ is the set of $e$ directed edges between the tasks that maintain a partial order among them. The partial order introduces precedence constraints, i.e. if edge $e_{i,j} \in E$, then task $v_j$ cannot start its execution before $v_i$ completes. Matrix $D$ of size $n \times n$ denotes the communication data size, where $d_{i,j}$ is the amount of data to be transferred from task $v_i$ to task $v_j$.

In a given task graph, a root node is called an entry task and a leaf node is called an exit task. In this study, the task graph is assumed to be with single-entry and single-exit. If there is more than one exit or entry task, a zero-cost pseudo exit or entry task with zero-cost edges is used to connect them.
3.2.2 Resource Graph

Heterogeneous Distributed Computing System is modelled by a resource graph $R = \{P, L\}$, where $P$ represents a set of $m$ real-time computing processors and $L$ represents a set of links between any pair of processors in $P$. The link between processor $p_i$ and $p_j$ is represented as $l_{ij}$. The processors are connected in fully connected topology to form a heterogeneous distributed computing system. The bandwidth (data transfer rate) of the links between different processors in a heterogeneous system may be different depending on the kind of the network. The data transfer rate is represented by an $m \times m$ matrix, $DTR_{m \times m}$. $W$ is a $n \times m$ computation cost matrix in which each $W_{ij}$ gives the Estimated Computation Time (ECT) to complete task $v_i$ on processor $p_j$. The ECT value of a task may be different on different processor depending on the processor’s computational capability. The communication cost between two processors $p_x$ and $p_y$, depends on the channel initialization at both sender processor $p_x$ and receiver processor $p_y$ in addition to the communication time on the channel. This is a dominant factor and can be assumed to be independent of the source and destination processors. The channel initialization time is assumed to be negligible. The communication cost of the edge $e_{i,k}$, which is time for transferring data from task $v_i$ (scheduled on processor $p_x$) to task $v_k$ (scheduled on processor $p_y$) is defined by Equation (3.1)

$$CC_{i,k} = d_{i,k} / DTR_{x,y}$$

(3.1)

$CC_{i,k} = 0$ when both the tasks $v_i$ and $v_k$ are scheduled on the same processor. The data transfer rate for each link is assumed to be 1.0 and hence communication cost and amount of data to be transferred will be the same.
The failure of a resource in the distributed computing system is assumed to follow a Poisson process and each resource is associated with a constant failure rate $\lambda$. It should be noted that modelling the failure of a resource by a Poisson process may not always coincide with the actual failure dynamics of the resource. It is experimentally shown by (Plank and Elwasif 1998) that such an assumption may still result in reasonably useful mathematical models. For mathematical tractability, failures of resources are assumed to be statistically independent. It is also assumed that, once a resource has failed, it remains in the failed state for the remainder of the execution of the application. These assumptions about failures are common to other studies by (Kartik and Murthy 1997, Dogan and Ozguner 2002, 2005, Dongarra et al 2007, Tang et al 2010) for analysing the reliability of computer systems.

3.3 IMPROVED RELIABLE HEFT ALGORITHM (IRHEFT)

The load difference on the processor is used to characterize the workload of the processor. The balanced workload across the processors in the heterogeneous computing system is critical for availability and user efficiency. The probability of failure of a processor is likely to increase, if the workload on it increases. So any complex application must be scheduled in the distributed computing environment in a balanced manner. The task must be evenly distributed across processors in order to decrease the probability of failure.

At each step of scheduling, we need to see whether the load is balanced or not. For that, the load difference factor on processors is to be calculated.

The load difference of the processor $p_j$ when task $v_i$ is to be scheduled on it is calculated for each of the task-processor pair as:
\[
LD(v_i, p_j) = \sum_{x=1}^{n} \gamma_{xj} w_{x,j} - \sum_{x=1}^{n} \sum_{y=1}^{m} \gamma_{xy} w_{x,y} \left( \frac{1}{m} \right) \tag{3.2}
\]

Where \( \gamma_{xj} = 1 \) if the task \( v_x \) is scheduled on \( p_j \) and \( \gamma_{xj} = 0 \) if the task \( v_x \) is not scheduled on processor \( p_j \).

**Task Prioritization**

1. Compute \( \text{rank}_{x} \) for all tasks by traversing upwards, starting from the exit task.
2. \( \text{rank}_{x}(v_i) = \overline{w_i} + \max_{\tau_{j} \in \tau_{out}(v_i)} \left( d_{in} + \text{dir} + \text{rank}_{x}(v_j) \right) \)
3. Sort the tasks in descending order of \( \text{rank}_{x} \)

**Processor Selection**

4. While there are unused tasks in the scheduling list
5. \hspace{1cm} Do
6. \hspace{1cm} Select the first task \( v_j \) from the scheduling list
7. \hspace{1cm} For each processor \( p_j \) in the HDCS
8. \hspace{1cm} Compute \( SF(v_i, p_j) = EFT_{l,j} + C(v_i, p_j) + LD(v_i, p_j) \)
9. \hspace{1cm} EndFor
10. Assign task \( v_i \) to processor \( p_j \) having minimum value of \( SF \)
11. Mark the task \( v_i \) as scheduled
12. EndDo

**Figure 3.1 IRHEFT algorithm with load difference and cost function**

The improved Reliable HEFT (IRHEFT) algorithm is given in Figure 3.1. The algorithm has the same two phases as HEFT: (1) Task prioritization and (2) Processor selection phase. The task prioritization step uses the upward rank procedure as in HEFT. The upward rank of the tasks in the DAG is calculated. The upward rank of a task \( v_i \) is recursively defined by
\[ \text{rank}_u(v) = \bar{w}_i + \max_{v \in \text{succ}(v)} \left( \frac{d_{i,j}}{\bar{d}_{tr}} + \text{rank}_u(v) \right) \]  

(3.3)

where \( \text{succ}(v) \) is the set of immediate successors of task \( v \), \( d_{i,j} \) is the data to be transferred from task \( i \) to task \( j \), \( \bar{w}_i \) is the average computation cost of task \( i \) and \( \bar{d}_{tr} \) is the average data transfer rate over all the links.

The nodes are level sorted. Then the nodes at each level are arranged in descending order of their upward ranks.

Let \( \text{EST}(v_i, p_j) \) and \( \text{EFT}(v_i, p_j) \) are the Earliest Start Time (EST) and Earliest Finish Time (EFT) of task \( v_i \) on processor \( p_j \), respectively. For the entry task \( v_1 \), \( \text{EST}(v_1, p_j) = 0 \), and for other tasks in the graph, the EST and EFT values are computed recursively, starting from the entry task as given by Equation (3.4) and (3.6). In order to compute the EFT of task \( v_i \), all immediate predecessor tasks of \( v_i \) must have been scheduled.

\[ \text{EST}(v_i, p_j) = \max \{ \text{avail}[j], f_i \} \]  

(3.4)

where \( f_i = \max(\text{AFT}(v_i + \text{CC}_{v_i})) \)  

(3.5)

\[ \text{EFT}(v_i, p_j) = \text{W}_g + \text{EST}(v_i, p_j) \]  

(3.6)

where \( v_i \in \text{pred}(v_i) \) is the set of immediate predecessor tasks of \( v_i \) and \( \text{avail}[j] \) is the earliest time at which processor \( p_j \) is ready for task execution. If \( v_k \) is the last assigned task on processor \( p_j \), then \( \text{avail}[j] \) is the time that processor \( p_j \) completed the execution of the task \( v_k \) and it is ready to execute another task. The max block in the
Equation returns the ready time, i.e., the time when all the data needed by 
$v_i$ has arrived at processor $p_j$.

After a task $v_i$ is scheduled on a processor $p_j$, the earliest start 
time and the earliest finish time of $v_i$ on processor $p_j$ is equal to the actual 
start time $\text{AST}(v_i)$ and the actual finish time $\text{AFT}(v_i)$ of task $v_i$, 
respectively. After all tasks in a graph are scheduled, the makespan of the 
schedule will be the actual finish time of the exit task $v_n$ and is defined as:

$$\text{makespan} = \text{AFT}(v_n)$$  \hspace{1cm} (3.7)

The definition of Selection Factor (SF) is important part of the 
processor selection phase in the HEFT algorithm. The SF is used to identify the 
matching processor for the particular task. EFT is the only parameter in 
determining the SF in the HEFT algorithm, whereas in IRHEFT, the two 
parameters namely, reliability (cost function) and the load difference factor 
among the processors are accounted for defining the SF, in addition to, the EFT. 
Every task in the ready list is taken and its selection factor (SF) is calculated as:

$$\text{SF}(v_i, p_j) = \text{EFT}_{i,j} + C(v_i, p_j) + \text{LD}(v_i, p_j)$$  \hspace{1cm} (3.8)

where $\text{EFT}_{i,j}$ denotes the Earliest Finish Time of the task 
$v_i$ on $p_j$. $C(v_i, p_j)$ is the cost function as defined by Dogan and Ozguner 
(2002) for the Reliable Dynamic Level Scheduling (RDLS) algorithm. The 
cost function implies the unreliability in scheduling. There are three such cost 
functions.

The reliability index of a schedule $S$ is calculated as

$$\text{rel}(S) = \sum_j \text{FT}_j \times \lambda_j$$  \hspace{1cm} (3.9)
where $\text{FT}_j$ denotes the finish time of all the task assigned to the $j^{th}$ processor and $\lambda_j$ denotes the failure rate of the $j^{th}$ processor.

**Cost Function 1**

As reliability of the processor on which a task is executing is crucial for the reliability of the application, the first cost function is defined to be the time that the application will lose due to the failure of resources. That is, the cost will be equal to the rate of failure of the processor. The time of failure of a processor is relative to finish time of the task scheduled on that processor and failure rate of the processor. The first cost function is defined to be:

$$C_1(i,j) = \lambda_j \times w_{i,j}$$  \hspace{1cm} (3.10)

where $\lambda_j$ is the failure rate of $p_j$, $w_{i,j}$ is the computation time of task $v_i$ on processor $p_j$.

**Cost Function 2**

The second cost function is based on the following heuristics: (1) If a task has several dependants, it should be scheduled on more reliable resources. Due to this, the cost increases. (2) If the duration of the execution of task is long, it should also be scheduled on more reliable resources. Here also the cost increases due to high computation time.

$$C_2(i,j) = w_r(v_i) \times (1 - R_t(w_{i,j})) \times (w_{i,j})$$  \hspace{1cm} (3.11)

where $w_{i,j}$ is the computation time of task $v_j$ on processor $p_j$. The term $R_t(w_{i,j})$ is defined as
\[ R_t(w_{i,j}) = 1 - \lambda_j w_{i,j} \]  

(3.12)

\( w_R(v_i) \) is the weight of reliability of task \( v_i \). The median execution time of task across all processors and estimated time to transmit all relevant data from task to its successors are added. This sum is normalized by the maximum of the sum obtained across all tasks and the normalized value plus one is assigned as \( w_R(v_i) \).

**Cost Function 3**

The third cost function is defined as the time the application will lose due to the failure of processors or the failure of network resources, used during the inter task communication between the task \( v_i \) and its immediate predecessors \( \text{pred}(v_i) \).

\[
C_3(i, j) = \max \left\{ \lambda_j w_{i,j}, \max_{v_k \in \text{pred}(v_i)} \left\{ \lambda_j f_{k,i} \right\} \right\}
\]  

(3.13)

where \( f_{k,i} \) denotes the time when the execution of task \( v_k \) has finished execution and includes the amount of time needed to transfer all relevant data from \( p_i \) on which \( v_k \) is executing to processor \( p_i \). This is calculated using Equation (3.5).

The processor, for which selection factor is minimum, is chosen as the processor on which task \( v_i \) should be scheduled. Since cost function implies unreliability in scheduling, it is added in the calculation of \( SF \). The load difference is termed positive, when the processor has a huge workload and it is termed negative, when the processor has not been used for a long time. When unreliability increases, \( SF \) increases and thereby task \( v_i \) will be scheduled on the processor that has maximum reliability. When load
difference decreases, \( s_f \) decreases and thereby task \( v_i \) will be scheduled on the processor that has lesser load.

The complexity of IRHEFT algorithm is \( O(n^2 \times m) \) where rank calculation takes \( O(n \times n) \), in allocation dynamic level calculation and cost calculation take \( O(n^2 \times m) \) and load difference calculation takes \( O(n^2 \times m) \).

### 3.4 EXPERIMENTAL RESULTS AND DISCUSSIONS

The proposed IRHEFT algorithm is evaluated using two types of task graphs namely: (a) Random task graphs and (b) Fast Fourier Transformation task graph.

#### 3.4.1 Randomly generated task graphs

In the first part of evaluation, task graphs are generated randomly with the following input parameters:

- Number of nodes (tasks) in the graph, \( V \). The value is assigned from the set of \( \{6,100,500,1000\} \)
- Shape parameter of the graph, \( \alpha \). The height of a DAG is randomly generated from a uniform distribution with mean equal to \( \sqrt{\frac{V}{\alpha}} \). The width for each level in a DAG is randomly selected from a uniform distribution with mean equal to \( \alpha \times \sqrt{V} \). \( \alpha \) is assigned value from the set \( \{0.2,0.4,1.0,1.5\} \)
- Out degree value for all nodes are randomly generated from a uniform distribution with mean equal to out degree.
- Communication to computation ratio, CCR. CCR is the ratio of the average communication cost to the average computation
cost. The computation intensive applications may be generated by CCR=0.1 and lesser, whereas data-intensive applications may be generated by higher values of CCR. The CCR is assigned from the set \{0.1, 1.0, and 10.0\}.

- Amount of data $D$, to be transmitted between the tasks and is stored in a matrix $D(n \times n)$.

- Average computation cost in the graph, ACC. This is the average time required to complete the task on all of the available processors. Computation costs are generated randomly from a uniform distribution with mean equal to ACC. The value of ACC is set to 20.

- Heterogeneity factor $\beta$. This value indicates the range percentage of computation cost on processors. A high percentage value causes a significant difference in task’s computation cost among the processors and a low percentage indicates that the expected execution time of a task is almost equal on any given processor in the system. The value is assumed to be 0.5.

- The computation cost $w_{i,j}$ of each task $v_i$ on each processor $p_j$ in the system is randomly set from the range:

$$\bar{w}_i \times \left(1 - \frac{\beta}{2}\right) \leq w_{i,j} \leq \bar{w}_i \times \left(1 + \frac{\beta}{2}\right) \quad (3.14)$$

where $\bar{w}_i$ is the average computational cost of $v_i$ across all processors. Average computational cost of a task is the average time required to complete the task on all the processors. The average computation cost $\bar{w}_i$...
of each task \( v_i \) in the DAG is selected randomly from a uniform distribution with range \([0, \text{ACC}]\).

An example task graph with the 6 nodes with \( \alpha = 1.5, \text{ACC}=50, \text{CCR}=0.1, \beta = 0.5 \) is shown in Figure 3.2(a). The computational cost matrix for 3 processors is shown in Figure 3.2(b).

### 3.4.2 Resource graph

The resource graph is a complete graph. The parameters needed are:

- Number of processors (m)
- Data Transfer Rate (DTR) in the links connecting the processors. To define the data transfer rate over all combinations of processors, the average DTR is considered and the value is assumed to be 1.0.
- Failure rate of the processors and links \( \lambda \). The values are assumed to follow Poisson distribution and lies in between \( 10^{-3} \) and \( 10^{-4} \).

The resource graph with three processors in a fully connected topology is shown in Figure 3.2 (c).
Figure 3.2(a) Task graph with n=6, α = 1.5, CCR=0.1, β = 0.5

<table>
<thead>
<tr>
<th></th>
<th>P₁</th>
<th>P₂</th>
<th>P₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>v₁</td>
<td>50</td>
<td>49</td>
<td>51</td>
</tr>
<tr>
<td>v₂</td>
<td>49</td>
<td>48</td>
<td>50</td>
</tr>
<tr>
<td>v₃</td>
<td>51</td>
<td>50</td>
<td>52</td>
</tr>
<tr>
<td>v₄</td>
<td>48</td>
<td>47</td>
<td>49</td>
</tr>
<tr>
<td>v₅</td>
<td>52</td>
<td>51</td>
<td>53</td>
</tr>
<tr>
<td>v₆</td>
<td>50</td>
<td>49</td>
<td>51</td>
</tr>
</tbody>
</table>

Figure 3.2(b) Computational cost matrix

3.4.3 Performance of IRHEFT for random task graphs

For random task graphs, the performance of the proposed algorithm varies with respect to application size and the CCR. For small task graphs with smaller values of CCR, the cost functions values are small as compared to the other terms in Equation (3.8). Hence, we have used CCR values of 1.0 and 10.0. The significance of the load difference factor in generating reliable schedules is studied by implementing the proposed IRHEFT algorithm without the load difference term in line 8 of Figure 3.1.
Random task graphs are generated and the makespan and reliability index are evaluated using RHEFT, RDLS, and IRHEFT with and without Load Difference factor. The reliability index is scaled and plotted for better readability and to show the decrease in the value achieved by the load balance IRHEFT.

The makespan and reliability index of random task graphs by the IRHEFT algorithm are compared with RHEFT and RDLS. The load difference factor is not introduced in scheduling decision in the IRHEFT algorithm for the comparison with RHEFT and RDLS in Figure 3.3 through Figure 3.6. Figure 3.3 shows that RHEFT achieves lesser makespan values compared to IRHEFT. The failure rate of the processors is considered in the processor selection phase of HEFT for implementing RHEFT. Hence, RHEFT generates lesser schedule lengths compared to IRHEFT.

![Figure 3.3 Makespan of RHEFT and IRHEFT without load difference for random task graphs with CCR=1.](image)

Figure 3.4 shows that as the size of the application increases, the performance of IRHEFT with cost functions $C_2$ and $C_3$ is identical. They produce lesser reliability index value and hence more reliability compared to RHEFT. The increase in reliability is due to the reason that the failure rate of the processors is also introduced in the finish time calculation of a task.
Figure 3.4 Reliability index of RHEFT and IRHEFT without load difference for random task Graphs with CCR=1

Figure 3.5 Makespan of RDLS and IRHEFT without load difference for random task graphs with CCR=1

Figure 3.5 shows the comparison between the IRHEFT and RDLS algorithms in terms of makespan. The makespan values of the random task graphs are lesser compared to that of RDLS with all the three cost functions. The reliability index values are better for the RDLS algorithm compared to IRHEFT and are shown in Figure 3.6.
Figure 3.6 Reliability index of RDLS and IRHEFT without load difference for Random Task Graphs with CCR=1

The incorporation of the $C(v_j,p_j)$ term alone on the scheduling decision in RDLS algorithm results in significant drops in reliability index at the expense of increasing makespan. To reduce the percentage of increase in makespan the load difference factor is introduced in the scheduling decision of IRHEFT and the performance is compared with RDLS in Figure 3.7 through Figure 3.10.

Figure 3.7 Makespan of RHEFT and IRHEFT with load difference for random task graphs with CCR=1
Figure 3.8 Reliability index of RHEFT and IRHEFT with load difference for random task graphs with CCR=1

Figure 3.9 Makespan of RDLS and IRHEFT with load difference for random task graphs with CCR=1
Figure 3.10  Reliability index of RDLS and IRHEFT with load difference for random task graphs with CCR=1

The percentage of improvement depends on the communication-to-computation ratio. Table 3.1 shows the percentage of increase in the makespan and the percentage of decrease in failure probability of random task graphs under RDLS as compared to DLS with and without the load difference factor. According to Table 3.1, for CCR=1.0, the failure probability of applications under RDLS with load difference decreases nearly at the same rate as the increase in execution time except for cost function $C_3$. For CCR=10.0, RDLS with load difference shows significant decrease in failure probability at a relatively lesser amount of increase in makespan. Table 3.1 compares the RHEFT and IRHEFT algorithms implemented with and without load difference factor for random task graphs.
Table 3.1 Comparison of DLS with RDLS in terms of failure probability with CCR values 1.0 and 10.0

<table>
<thead>
<tr>
<th>CCR</th>
<th>Cost Function</th>
<th>Without Load Difference</th>
<th>With Load Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Increase in Makespan</td>
<td>Decrease in Failure Probability</td>
</tr>
<tr>
<td>1.0</td>
<td>1</td>
<td>36%</td>
<td>2.6%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>39%</td>
<td>2.8%</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>55%</td>
<td>4.8%</td>
</tr>
<tr>
<td>10.0</td>
<td>1</td>
<td>12%</td>
<td>0.7%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>37.8%</td>
<td>13%</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>70%</td>
<td>38%</td>
</tr>
</tbody>
</table>

Table 3.2 reveals that IRHEFT algorithm with cost function $C_2$ outperforms the algorithm with the cost functions $C_1$ and $C_3$ in terms of minimizing the failure probability at the price of lesser increase in makespan.

Table 3.2 Comparison of RHEFT with IRHEFT in terms of failure probability with CCR values 1.0 and 10.0

<table>
<thead>
<tr>
<th>CCR</th>
<th>Cost Function</th>
<th>Without Load Difference</th>
<th>With Load Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Increase in Makespan</td>
<td>Decrease in Failure Probability</td>
</tr>
<tr>
<td>1.0</td>
<td>1</td>
<td>10%</td>
<td>0.7%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>29%</td>
<td>2.0%</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>45%</td>
<td>2.7%</td>
</tr>
<tr>
<td>10.0</td>
<td>1</td>
<td>5%</td>
<td>0.3%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>20%</td>
<td>6%</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>61%</td>
<td>32%</td>
</tr>
</tbody>
</table>
3.4.4 Performance of IRHEFT for real application task graphs

The proposed algorithm is tested with the recursive, one-dimensional Fast Fourier Transformation (FFT) task graph. This was studied by (Chung and Ranka 1992, Topcuoglu et al 2002). The recursive algorithm and the structure of FFT graph is shown in the Figure 3.11(a) and 3.11(b) respectively. In Figure 3.11(a) M is an array of size m which holds the co-efficients of the polynomial and array Y is the output of the algorithm. The algorithm consists of two parts: recursive calls (lines 3-4) and butterfly operations (lines 6-7). The task graph can be divided into two parts: the tasks above the dashed line are recursive call tasks and the ones below the line are butterfly operation tasks. For an input vector of size m there are $2^m - 1$ recursive call tasks and $m \times \log_2 m$ butterfly operation tasks. We assume $m = 2^k$ for some integer k. The number of data points in FFT is another parameter for simulation, which varies from 4 to 64 incrementing by power of 2. The values of CCR and range parameters are same as that for random task graphs.

\[
\begin{align*}
\text{FFT}(M, \omega) \\
1. & \quad n = \text{length}(M) \\
2. & \quad \text{if } (n = 1) \text{return } (M) \\
3. & \quad Y^{(0)} = \text{FFT}(M[0], M[2], \ldots M[n-2], \omega^2) \\
4. & \quad Y^{(1)} = \text{FFT}(M[1], M[3], \ldots M[n-1], \omega^2) \\
5. & \quad \text{for } i = 0 \text{ to } n/2 - 1 \text{ do} \\
6. & \quad Y[i] = Y^{(0)}[i] + \omega^i \times Y^{(1)}[i] \\
7. & \quad Y[i + n/2] = Y^{(0)}[i] - \omega^i \times Y^{(1)}[i] \\
8. & \quad \text{return } (Y)
\end{align*}
\]

**Figure 3.11(a) Fast Fourier Transformation algorithm**
Figure 3.11(b) Structure of FFT graph with data points d=4

Figure 3.12(a) and (b) shows the performance of the proposed IRHEFT algorithm with the cost function $C_2$. The inclusion of the load difference factor has resulted in considerable decrease in reliability index at the lesser increase in makespan.

Figure 3.12.(a) Comparison of Makespan values for FFT graphs scheduled by IRHEFT C2 with and without load difference
3.5 CONCLUSION

In this chapter, the problem of scheduling tasks on heterogeneous environment has been studied. The problem deals with two objectives: maximizing reliability and minimizing makespan. As these two objectives are unrelated and sometime contradictory, we need to search for bi-objective optimisation algorithms.

The two previously proposed algorithms namely, RHEFT and RDLS efficiently minimize makespan and maximises reliability, respectively. An improved algorithm (IRHEFT) that combines the features of HEFT and RDLS for solving the task scheduling problem has been proposed. This improved algorithm extends the traditional list scheduling heuristics to optimize both the objectives simultaneously. In addition IRHEFT includes the load difference factor to improve the reliability. The performance of the algorithm is compared with the existing bi-objective RHEFT and RDLS algorithms. The comparison shows that there is a significant increase in the reliability factor due to the introduction of load difference factor in the processor selection phase. Thus the two objectives of maximizing the reliability and minimizing the schedule length have been achieved by our proposed algorithm.