CHAPTER 6

LOAD BALANCING AND FAULT TOLERANT SCHEDULING ALGORITHM WITH USER SATISFACTION

This chapter proposes a new scheduling algorithm for computational grids that considers load balancing, fault tolerance and user satisfaction based on the grid architecture, resource heterogeneity, resource availability and task characteristics such as user deadline.

6.1 INTRODUCTION

The main aspect of grid computing considered in this chapter is the dynamicity of resources. The resources of grid can be freely added or withdrawn at any time according to the owner’s discretion. So, the performance of grid nodes and their load frequently changes with respect to time (Torkestani 2011). Load balancing attempt to improve the response time and also ensure maximum utilization of resources available. Load balancing can be defined by certain policies such as information policy, triggering policy, resource type policy, location policy and selection policy (Karatza 2001).

Load balancing algorithms can be classified into two categories such as static algorithms and dynamic algorithms. In static load balancing, decision is made at compile time when resource requirements are estimated. Dynamic load balancing algorithm allocates/reallocates resources at runtime. The second aspect considered in this research is fault tolerance which deals
with failure of resources dynamically. When a task is submitted to a grid broker, it schedules and allocates it to the selected resource. If that resource fails due to any of the following reasons, that should also be handled by the proposed algorithm. Failures may be network failure, resource failure etc. There are two types of failure handling mechanisms such as proactive and passive. In proactive mechanisms, the failure consideration for the grid is made before scheduling a task and dispatched with hopes that the task does not fail. Whereas, passive mechanisms handles the task failures after it has occurred. However, in the dynamic systems only post-active mechanism is relevant (Medeiros et al 2003).

The third aspect is user satisfaction which is achieved by considering the user deadline of the tasks submitted. The tasks are received along with its deadline. The task is expected to be executed within the deadline.

The main contribution of this chapter is

- An algorithm that aims at achieving balanced load along with fault tolerance and user satisfaction is introduced for scheduling independent tasks on computational grid resources.

- The performance of the Load Balancing and Fault Tolerant (LBFT) scheduling algorithm with user satisfaction is compared with an existing fault tolerant load balancing algorithm and another algorithm that focuses on user satisfaction.

For evaluating the performance of the LBFT scheduling algorithm, a hybrid load balancing algorithm proposed by Balasangameshwara & Raju (2012), Min-min, FTMM and BSA algorithms are compared. The rest of this
chapter provides a detailed discussion of the materials and methods proposed to achieve the objectives.

6.2 HYBRID LOAD BALANCING ALGORITHM (AlgHybrid_LB)

In this section, a brief description of AlgHybrid_LB is provided. This algorithm considers both load balancing and fault tolerance factors for scheduling independent tasks. The load balancing policy adopted is a hybrid load balancing policy which initially adapts to static mode and after submission of tasks to resources; it adjusts dynamically until it reaches a balance state (Balasangameshwara & Raju 2012). Dynamic load balancing is adopted only when a resource becomes ineffective and not with an effective resource. It also uses a passive fault tolerant methodology having backups for all tasks. For load balancing, the job queue length is taken as the load indicator. Threshold value is calculated for each site and an effective site will have many neighbour sites which the scheduler will use to select an effective neighbouring site for offloading tasks. For each effective site, other neighbour sites are sorted by quantified transmission rate in ascending order. After this process, the first ranked site is chosen as the nearest site.

It uses first come first served algorithm for allocating tasks to sites. It follows a load adjustment policy that tries to reduce the load difference among a site and its neighbouring sites by migrating tasks from heavily loaded to lightly loaded sites. Its efficiency is compared with MCT and Min-min algorithm and it is proved that the completion time is less than Min-min and MCT algorithms.

6.3 LBFT SCHEDULING MODEL

The LBFT scheduling model is given in Figure 6.1. A load balancing decision making module is included which performs load calculation of resources and categorizes them as overloaded, underloaded and
normally loaded. With this information, the scheduler schedules the task to resources based on threshold value.

![Figure 6.1 LBFT scheduling model](image)

### 6.4 LBFT ALGORITHM

Grid scheduler receives the tasks with user deadline $UD(T_i)$. The task’s information such as its length in MI is used to calculate the execution time $ETC(T_i, R_j)$ of each task in each of the available resources. With the ready time information $RT(R_j)$ available for each resource at GIS, the algorithm calculates the completion time $CT(T_i, R_j)$. The failure information of resources such as number of tasks submitted to a resource $T_{sub}$ and number of tasks successfully completed $T_{succ}$ and number of tasks not completed successfully $T_{f}$ is also available in GIS which helps in calculating the failure rate $FR(R_j)$. 
Since the main focus is on load balancing, calculation of load of each PE, machine and resource becomes essential.

Hence the load of PE is given by Equation (6.1),

$$\text{Load}(PE_i) = \frac{\sum_{i=0}^{n} M_i}{MIPS_i \times AT_i}$$ \hspace{1cm} (6.1)

where \(m\) is the number of tasks allocated to \(PE_i\) and \(AT_i\) is the availability time of \(PE_i\). Machine is the collection of PE’s and the average load of each machine \(M_i\) is calculated using the Equation (6.2),

$$\text{AL}(M_i) = \frac{\sum_{k=1}^{n} \text{Load}(PE_k)}{n}$$ \hspace{1cm} (6.2)

where \(n\) is the number of PE’s under machine \(i\). Resource is the collection of machines and the average load of each resource \(R_i\) is calculated by the Equation (6.3),

$$\text{AL}(R_i) = \frac{\sum_{k=1}^{n} \text{AL}(M_k)}{n}$$ \hspace{1cm} (6.3)

where \(n\) is the number of machines under resource \(i\). The average load of the system is calculated in Equation (6.4),

$$\text{AL} = \frac{\sum_{k=1}^{n} \text{AL}(R_k)}{n}$$ \hspace{1cm} (6.4)

where \(n\) is the number of resources in the system. In order to balance the load of the resources in grid, a balance threshold \(\Omega\) is defined such that the average loads of each resource \(\text{AL}(R_i)\) to be less than the balance threshold of the system and is given in Equation (6.5),

$$\Omega = AL + \sigma$$ \hspace{1cm} (6.5)
where $\sigma$ is the deviation factor or standard deviation of the load of the system which is defined in Equation (6.6),

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(AL(R_i) - AL)^2}{N}} \quad (6.6)$$

where $N$ is the number of resources in the system. The new load of PE can be calculated by using the Equation (6.7).

$$NewLoad(P_{E_j}) = Load(P_{E_j}) + \frac{M_{I_i}}{MIPS_j \times AT_j} \quad (6.7)$$

The list of resources in which the task gets completed within user deadline is collected for each task and they are sorted based on their failure rate. Based on the balance threshold, the resources are categorized as overloaded and underloaded and finally the load is balanced by submitting the task to the underloaded resource. When a resource is assigned a task, the load of each resource and system, balance threshold, failure rate and ready time are recalculated. The same procedure is repeated for all tasks till the task list becomes empty.
Step 1: Get the Task_list $T$ of tasks submitted by the user with its user deadline $UD(T_i)$

Step 2: Get the list $R$ of resources available in grid from GIS and initialize the counters Deadline Hit Count and Hit Count

Step 3: Construct $ETC(T_i, R_j)$ matrix of size $m \times n$ where $m$ represents the number of tasks and $n$ represents the number of resources involved.

Step 4: For all resources $R_j$ in $R$, where $1 \leq j \leq n$, and $n$ denotes number of resources,
   do
   \begin{enumerate}
   \item Calculate Failure rate $FR(R_j)$
   \item Calculate Ready Time $RT(R_j)$
   \item Calculate Load of each Processing Element, Average Load of each machine and Average Load of each resource
   \end{enumerate}
   done

Step 5: Calculate Average Load of the system and Balance Threshold

Step 6: Create a list of underloaded resources $UR$ which has $AL(R_j) < \Omega$

Step 7: For each task in $T_i$ in queue and for each resource $R_j$.
   do
   \begin{enumerate}
   \item Construct $CT(T_i, R_j)$, $DT(T_i, R_j)$, $TCT(T_i, R_j)$ matrix of size $m \times n$
   \end{enumerate}
   done

Step 8: For all task $T_i$ in Task_list $T$,
   do

Figure 6.2 (Continued)
8.1: Create lists $UT_{i1}$ and $UT_{i2}$ with resources that has
$TCT(T_i, R_j) \leq UD(T_i)$ and $TCT(T_i, R_j) > UD(T_i)$
respectively.

8.2: Sort the lists $UT_{i1}$ and $UT_{i2}$ based on $FR(R_j)$ of resources
in ascending order

8.3: Create lists $ULT_{i1}$ and $ULT_{i2}$ with the set of underloaded
resources from $UT_{i1}$ and $UT_{i2}$ respectively in order.

8.4: If entries in $ULT_{i1}$,
    Select the first resource in the list for task $T_i$
    and check for load balancing using the equation 6.7 and if the load is
    balanced, dispatch $T_i$ to resource $R_j$ and Increment Deadline hit count and
    Hit count. Otherwise, select next resource and repeat step 8.4.
    else if entries in $ULT_{i2}$,
    Select the first resource in the list for task $T_i$
    and check for load balancing using the equation 6.7 and if the load is
    balanced, dispatch $T_i$ to resource $R_j$ and Increment Hit Count. Otherwise,
    select next resource and repeat step 8.4.

8.5: Remove task $T_i$ from Task_list $T$.
8.6: Update $RT(R_j)$ and $FR(R_j)$ where $j$ is the resource to
    which the task $T_i$ is dispatched.

done

Step 9: If there are tasks in Task_list $T$,
    Repeat steps from 4.3.
else
    Compute Makespan, Hit count, Deadline hit count and
    Average resource utilization.
endif

Figure 6.2 LBFT scheduling algorithm
6.5 EXPERIMENTAL ENVIRONMENT

The simulation is done by using gridsim simulator with enhanced features for fault tolerant scheduling. The number of resources considered is 16 and the number of tasks considered for a batch is 512.

6.6 EVALUATION

The main aim of the proposed scheduling algorithm is to minimize the makespan and to improve fault tolerance of the system proactively with balanced load of resources. Fault tolerance is achieved by increasing the Hit count. User Satisfaction is achieved by improving the deadline hit count. Load balancing is achieved by improving average resource utilization. Similar to the newly proposed algorithms such as FTMM and BSA, LBFT also follows task and resource heterogeneity with ETC matrix calculations.

6.6.1 Performance Metrics

The performance metrics used to evaluate the proposed algorithm are defined below.

Makespan, Hit count, Deadline hit count: These metrics are already defined in the previous chapters in section 4.7.1 and 5.7.1.

Average resource utilization: This metric is newly introduced in order to measure the load balancing which can be calculated as follows.

The utilization of each resource $RU(R_j)$ can be calculated by the Equation (6.8).

$$RU(R_j) = \frac{\sum_{i=0}^{\infty} M_i}{\text{MIPS}_j \times AT_j} \times 100 \quad (6.8)$$
The average resource utilization $ARU$ of the system can be calculated using Equation (6.9).

$$ARU = \frac{1}{N} \sum_{i=1}^{N} RU(R_i)$$  \hspace{1cm} (6.9)

where $N$ is the number of resources.

6.7 EXPERIMENTAL RESULTS

The LBFT algorithm is evaluated for all the four metrics defined in the previous section. It is compared with Min-min algorithm, FTMM, BSA algorithms and a hybrid load balancing algorithm (AlgHybrid_LB) for load balancing and fault tolerance.

6.7.1 Makespan

The results of makespan for LBFT algorithm is compared with Min-min, FTMM, BSA and AlgHybrid_LB algorithm and is shown below in Figure 6.3. Even though LBFT considers load balancing, fault tolerance and user deadline, the makespan is reduced than Min-min, BSA and FTMM algorithms.

![Figure 6.3 Makespan of Min-min, FTMM, BSA, AlgHybrid_LB & LBFT](image-url)
The AlgHybrid_LB algorithm proposed for load balancing is also compared with LBFT and the results show that LBFT has considerable decrease in makespan than AlgHybrid_LB.

6.7.2 Hit Count

The results of hit count for LBFT algorithm is compared with Min-min, FTMM, BSA and AlgHybrid_LB algorithm and is shown in Figure 6.4. The results show that LBFT has more number of tasks successfully completed without any resource failure.

![Figure 6.4 Hit count of Min-min, FTMM, BSA, AlgHybrid_LB & LBFT](image-url)
6.7.3 Deadline Hit Count

![Figure 6.5 Deadline hit count of Min-min, FTMM, BSA, AlgHybrid_LB & LBFT](image)

The results of deadline hit count for LBFT algorithm is compared with Min-min, FTMM, BSA and AlgHybrid_LB algorithm and is shown in Figure 6.5 and the results show that LBFT has relatively high deadline hit count than Min-min, FTMM. Since BSA is specifically designed for user satisfaction and LBFT concentrates on load balancing, LBFT has high deadline hit count in all cases. When compared with AlgHybrid_LB, LBFT has highest hit count than AlgHybrid_LB.

6.7.4 Average Resource Utilization (ARU)

The average resource utilization of Min-min, FTMM, BSA, AlgHybrid_LB and LBFT are presented in Figure 6.6. The results show that the average resource utilization of LBFT is higher than the other algorithms.
Figure 6.6 ARU of Min-min, FTMM, BSA, AlgHybrid_LB & LBFT

6.8 PERFORMANCE ANALYSIS

6.8.1 Makespan

The percentage improvement of LBFT over Min-min, FTMM, BSA and AlgHybrid_LB based on makespan is shown in Table 6.1.

<table>
<thead>
<tr>
<th>CASES</th>
<th>Improvement over Min-min (%)</th>
<th>Improvement over FTMM (%)</th>
<th>Improvement over BSA (%)</th>
<th>Improvement over AlgHybrid_LB (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>24.4</td>
<td>19.4</td>
<td>12.1</td>
<td>19.6</td>
</tr>
<tr>
<td>2</td>
<td>21.1</td>
<td>6</td>
<td>5.7</td>
<td>15.5</td>
</tr>
<tr>
<td>3</td>
<td>20.7</td>
<td>12.6</td>
<td>1.3</td>
<td>17.9</td>
</tr>
<tr>
<td>4</td>
<td>48.6</td>
<td>36.6</td>
<td>15.9</td>
<td>46.2</td>
</tr>
</tbody>
</table>
The average percentage improvement of LBFT based on makespan towards Min-min is 30.1 %. With FTMM, the percentage improvement is 18.4 % and towards BSA, an improvement of 7.6% is achieved. When compared with AlgHybrid_LB, it shows an average percentage improvement of 26.5%. Hence, it is notable that LBFT has reduced makespan than the FTMM, Min-min, BSA and AlgHybrid_LB algorithms.

6.8.2 Hit Count

The percentage improvement of LBFT over Min-min, FTMM, BSA and AlgHybrid_LB based on hit count is shown in Table 6.2. The average percentage improvement of LBFT based on hit count towards Min-min is 26.3%. With FTMM, the percentage improvement is 11.8 % and towards BSA, an improvement of 3.4% is achieved. When compared with AlgHybrid_LB, it shows an average percentage improvement of 5.7%. Hence, it is notable that LBFT has more hit count than the FTMM, Min-min, BSA and AlgHybrid_LB algorithms.

Table 6.2 Percentage improvement of LBFT based on Hit Count

<table>
<thead>
<tr>
<th>CASES</th>
<th>Improvement over Min-min (%)</th>
<th>Improvement over FTMM (%)</th>
<th>Improvement over BSA (%)</th>
<th>Improvement over AlgHybrid_LB (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25.1</td>
<td>17.2</td>
<td>2.1</td>
<td>2.9</td>
</tr>
<tr>
<td>2</td>
<td>32.6</td>
<td>11.2</td>
<td>3.9</td>
<td>9.6</td>
</tr>
<tr>
<td>3</td>
<td>22.3</td>
<td>12.9</td>
<td>3.7</td>
<td>1.3</td>
</tr>
<tr>
<td>4</td>
<td>25.2</td>
<td>5.9</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>
6.8.3 Deadline Hit Count

The percentage improvement of LBFT over Min-min, FTMM, BSA and AlgHybrid_LB based on deadline hit count is shown in Table 6.3.

Table 6.3 Percentage improvement of LBFT based on Deadline Hit Count

<table>
<thead>
<tr>
<th>CASES</th>
<th>Improvement over Min-min (%)</th>
<th>Improvement over FTMM (%)</th>
<th>Improvement over BSA (%)</th>
<th>Improvement over AlgHybrid_LB (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>38</td>
<td>31.2</td>
<td>2.7</td>
<td>33.6</td>
</tr>
<tr>
<td>2</td>
<td>36.7</td>
<td>30.5</td>
<td>1.4</td>
<td>52.7</td>
</tr>
<tr>
<td>3</td>
<td>40.6</td>
<td>33.9</td>
<td>5.5</td>
<td>14.5</td>
</tr>
<tr>
<td>4</td>
<td>14.7</td>
<td>6.8</td>
<td>5.4</td>
<td>16.4</td>
</tr>
</tbody>
</table>

The average percentage improvement of LBFT based on deadline hit count towards Min-min is 32.5%. With FTMM, the percentage improvement is 25.6% and towards BSA, an improvement of 3.8% is achieved. When compared with AlgHybrid_LB, it shows an average percentage improvement of 29.3%. Hence, it is notable that LBFT has better user satisfaction than the FTMM, Min-min, BSA and AlgHybrid_LB algorithms.

6.8.4 Average Resource Utilization

The average percentage improvement of LBFT based on average resource utilization towards Min-min is 18.8%. With FTMM, the percentage improvement is 12.5% and towards BSA, an improvement of 10.3% is achieved. When compared with AlgHybrid_LB, it shows an average
percentage improvement of 9%. Hence, it is notable that LBFT has better resource utilization than the FTMM, Min-min, BSA and AlgHybrid_LB algorithms.

6.9 CONCLUSION

A scheduling algorithm for load balancing along with user satisfaction and fault tolerance is proposed and the performance is evaluated based on makespan, hit count, deadline hit count and resource utilization. The results and performance analysis shows that the proposed LBFT improves system performance by reduced makespan, increased hit count and deadline hit count and increased resource utilization. The next chapter gives a detailed description of a hierarchical scheduling algorithm with load balancing, fault tolerance and user satisfaction.