CHAPTER 4

DESIGN OF PRIORITIZED USER DEMAND APPROACH FOR META TASKS

A new prioritized user demand algorithm is presented in this section which mainly focuses on the deadlines of the available static jobs as expected by the users. User demand aware scheduling algorithm for data intensive tasks is also proposed which considers the data transfer requirements along with user deadline of the jobs. The PUD algorithm mainly concentrates on user demand and computational time of jobs on the resources. The UDDA algorithm includes data transfer time required for the jobs.

4.1 INTRODUCTION

Grid scheduling is the process of scheduling tasks over grid resources. Grid scheduler acts as a medium to receive jobs from various users and allocate them to appropriate resources. An efficient scheduling algorithm must improve the overall system performance by reducing waiting time for the individual task and makespan for the set of jobs.

There are two main factors that are considered in this scheduling algorithm, namely, system performance and user satisfaction. PUD and UDDA algorithms are both system centric and application centric. The algorithms which are proposed in this section mainly deal with the statically available jobs and hence adopt the static scheduling mode. They deal with a list of jobs at a time and have two phases in scheduling such as job
prioritizing and resource selection. The job prioritizing phase sets the priority of each job based on the user deadline as the parameter and generates a scheduling list by sorting the jobs according to user deadline and expected execution time. The resource selection phase selects jobs in the order of their priorities and maps each selected job to its optimal resource. In the second phase, an optimal resource is selected for each job based on whether the resource will complete the job within the user deadline or not. If the resource satisfies the criteria, then it will be allocated to the job.

4.2 SCHEDULING MODEL

Figure 4.1 shows the grid scheduling model of the proposed system. The four basic building blocks of the grid scheduling architecture are user, resource broker, Grid Information Service (GIS) and the resources. User submits jobs to the scheduler which is responsible for selecting the suitable resource for executing the job.

![Diagram of scheduling architecture used in PUD and UDDA](image)

Figure 4.1 Scheduling architecture used in PUD and UDDA

Grid information service has information about the resources like capability, cost for execution and baud rate. The scheduler identifies
appropriate resources based on job requirements like user deadline, job length and size of data.

An effective grid scheduler should reduce the waiting time and makespan of the jobs by utilizing the resources efficiently. It consists of two phases: job scheduling and resource allocation. Job scheduling is the process of selecting a job from the job pool that is to be executed next in a grid resource. Resource allocation is the process of allocating a resource for completing the job.

Scheduling algorithm can reduce the makespan and waiting time by considering the transfer rate of data and communication overhead. A centralized broker is the single point for the whole infrastructure and manages directly the resource manager interfaces that interact directly with LRM. All users submit the tasks to the centralized broker. Each resource differs from other resources by many ways that includes number of processing elements, processing speed, internal scheduling policy, load factor etc. Similarly each job differs from other jobs by length, deadline, time zone etc.

4.3 SCHEDULING ALGORITHMS

This section describes two algorithms for scheduling the tasks. First, the PUD algorithm considers user deadline and computation time of the job at the resource. Second, UDDA algorithm for data intensive tasks considers user deadline, computation time of the job at the resource and data transfer time to the resource.

4.3.1 Prioritized User Demand Algorithm

This algorithm is based on user satisfaction and system performance. It takes user’s deadlines into account and makes the job to be
executed within the expected deadline by assigning it to the most suitable resource. It also concentrates on the system performance by reducing the idle time of the resources and distributing the unmapped jobs equally among the available resources. Initially, it calculates the Expected Time to Compute (ETC) matrix and concentrates on the completion time and hence the system’s performance is also a major consideration in addition to user’s satisfaction. This scheduling process has two major steps. In the first step, jobs are prioritized based on the user demand (user deadline) and the second step is based on the system performance (completion time of the job). The symbols used in PUD and UDDA algorithms are listed in Table 4.1.

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{ECT}_{i,j}$</td>
<td>Expected completion time of the job $i$ in resource $j$</td>
</tr>
<tr>
<td>$\text{RT}_j$</td>
<td>Ready time of resource $j$</td>
</tr>
<tr>
<td>$\text{EET}_{i,j}$</td>
<td>Expected execution time of the job $i$ in resource $j$</td>
</tr>
<tr>
<td>$\text{DT}_{i,j}$</td>
<td>Data transfer time for the job $i$ to the resource $j$</td>
</tr>
<tr>
<td>$\text{DF}_{i,j}$</td>
<td>Difference Factor for job $i$ at the resource $j$</td>
</tr>
<tr>
<td>$\text{UT}_i$</td>
<td>User time or deadline given by the user for job $i$</td>
</tr>
<tr>
<td>$\text{MIPS}_j$</td>
<td>Processing capacity or speed of resource $j$</td>
</tr>
<tr>
<td>$\text{MI}_i$</td>
<td>Length of job $i$</td>
</tr>
</tbody>
</table>

Expected Execution Time of job or task $i$ in resource $j$ ($\text{EET}_{i,j}$) is based on the size of the job, MIPS rate of the resource and it is calculated as follows.

$$\text{EET}_{i,j} = \frac{\text{MI}_i}{\text{MIPS}_j} \quad (4.1)$$
Ready Time of resource $j$ ($RT_j$) is based on the jobs that are already available at the resource and it is calculated as follows.

$$RT_j = \frac{\sum_{i=1}^{n} MI_i}{MIPS_j} \tag{4.2}$$

Expected Completion Time of job $i$ in resource $j$ ($ECT_{ij}$) is summation of ready time of the resource and expected execution time of job $i$ in resource $j$ and it is calculated as follows.

$$ECT_{ij} = EET_{ij} + RT_j \tag{4.3}$$

PUD scheduling algorithm is given in Algorithm 4.1. In this approach, users submit tasks to the scheduler with information such as job length, user deadline, etc. The scheduler collects information about resources. Then, expected completion time for all the jobs in all the resources i.e. the time to complete a job when the job is assigned to a resource is calculated. Then the jobs are prioritized based on the user deadline of the jobs. After prioritizing the jobs, the jobs are selected based on their priority from the unassigned list which contains the set of jobs to be executed. Then the difference factor ($DF_{ij}$) is calculated based on user deadline and expected completion time for all job and resource pairs. Then the resource which has minimum difference factor is selected for the job and the job is assigned to the resource. After assigning the job to the resource, the job is removed from the unassigned list. Then, the waiting time of the resource is updated and these steps are performed until all the jobs are scheduled. Thus both user satisfaction and system performance are taken into consideration effectively in this algorithm.
Algorithm 4.1 PUD algorithm

MAX\_VALUE=1000

While there are jobs to be allocated do

For each job \( j_i \) to be scheduled, \( j_i \in J, 1 \leq i \leq n \),
do

For each resource \( r_j \) available, \( r_j \in R, 1 \leq j \leq m \),
do

Calculate \( \text{ECT}_{i,j} = \text{EET}_{i,j} + \text{RT}_j \)
end for
end for

For each job \( j_i \) in \( J \)
do

Select a job \( j_i \) which has minimum \( UT \) where \( 1 \leq i \leq n \)
For each resource \( r_j \in R \)
do

\( \text{DF}_{i,j} = \text{UT}_i - \text{ECT}_{i,j} \)
end for
if \( \text{DF}_{i,j} < 0 \), then \( \text{DF}_{i,j} = \text{MAX\_VALUE} \)

Allocate the job \( j_i \) to the resource \( r_j \) which has minimum difference factor value
Remove \( j_i \) from \( J \)
Update the ready time of the resource \( r_j \)

End for

End while
4.3.2 User Demand Aware Algorithm for Data Intensive Applications

For computation intensive tasks, the data transfer time is negligible. But, an efficient scheduling algorithm for data intensive tasks must consider the data transfer requirements of the jobs along with user requirements and time for job execution.

Since resources in grid environment are geographically distributed, the data transfer time is very much essential when a job is scheduled to the resource which is far from the location where the job is submitted. Data transfer time is based on the size of the data to be transferred for job \( i \) \((S_i)\) and baud rate of the resource \( j \) \((B_j)\). Data transfer time is calculated using the following formula.

\[
DT_{i,j} = \frac{S_i}{B_j}
\]  

(4.4)

A static heuristic approach is proposed for scheduling independent tasks in grid environment. The requirements of tasks are necessary to identify resources such as computational nodes and data resources. This scheduling algorithm considers both system and application aspects to improve the performance and utilization of the resources and throughput.

The UDDA algorithm given in Algorithm 4.2 considers user deadline for each job which is submitted by the user, data transfer time, waiting time and computation time for each \(<job, resource>\) pair. The job is selected from the job queue based on the user deadline of jobs and the resource is selected for each job based on the expected completion time of job.
In this algorithm, Expected Completion Time (ECT\textsubscript{i,j}) is the sum of ready time of the resource, expected execution time of the job and data transfer time for the job. It is calculated as follows.

\[ ECT\textsubscript{i,j} = RT\textsubscript{j} + EET\textsubscript{i,j} + DT\textsubscript{i,j} \]  \hfill (4.5)

MAX\_VALUE=1000
While there are jobs to be allocated do
    For each job \( j_i \) to be scheduled, \( j_i \in J, 1 \leq i \leq n \), do
        For each resource \( r_j \) available, \( r_j \in R, 1 \leq j \leq m \), do
            Calculate \( ECT\textsubscript{i,j} = RT\textsubscript{j} + EET\textsubscript{i,j} + DT\textsubscript{i,j} \)
        end for
    end for
    For each job \( j_i \) in J do
        Select a job \( j_i \) which has minimum UT where \( 1 \leq i \leq n \)
        For each resource \( r_j \in R \) do
            \[ DF\textsubscript{i,j} = UT\textsubscript{i} - ECT\textsubscript{i,j} \]
        end for
        if \( DF\textsubscript{i,j} < 0 \), then \( DF\textsubscript{i,j} = MAX\_VALUE \)
        Allocate the job \( j_i \) to the resource \( r_j \) which has minimum difference factor value
        Remove \( j_i \) from J
        Update the ready time of the resource \( r_j \)
    end for
end while

Algorithm 4.2 UDDA algorithm
4.4 SIMULATION RESULTS AND PERFORMANCE ANALYSIS

4.4.1 GridSim Toolkit

Grid computing involves millions of heterogeneous resources scattered across multiple organizations, different administrative domains and policies. The management of resources and scheduling of jobs in such an environment is complex. In order to evaluate the performance of resource management and job scheduling algorithms, there is requirement for sophisticated tools to be applied to real systems.

GridSim is a software platform which is flexible, modular and serves as a universal simulation toolkit for grid environment. It is based on event simulation library called SimJava. It provides functionality to simulate the basic characteristics of grid environment. It also provides the mechanism to represent common entities such as computational resources, jobs, network topology, data storage and other useful functionality (Buyya & Murshed 2002).

Grid entities can be extended to fit more complex requirements. This can be done by implementing a new java class which can be inherited from the existing GridSim class. Additional functions can then be implemented and new parameters can be added to provide the desired functionality of this new entity. Since all the entities like resource, job, etc., are defined as separate classes, it is easy to modify or create a new one.

GridSim toolkit provides modeling and simulation of a wide range of heterogeneous resources such as single or multiprocessors, shared and distributed memory machines such as PCs, workstations and clusters managed by time or space-shared schedulers. It can be used for modeling and simulation of application scheduling on various classes of parallel and distributed computing systems such as clusters and grids.
GridSim toolkit has the following salient features.

- It allows modeling of heterogeneous resources
- Resources can be modeled to operate under space or time shared mode
- Resources can be reserved for advance reservation
- Applications with different parallel models can be simulated
- Tasks can be heterogeneous and they can be computational or data intensive
- There is no limit on the number of tasks that can be submitted to a resource
- Network speed between resources can be specified
- It supports simulation of both static and dynamic schedulers
- Multiple user entities can submit their tasks for execution simultaneously to a resource which may be time-shared or space-shared
- New allocation policy can be made and integrated with the GridSim Toolkit.

4.4.2 PUD Algorithm

The experimental results are based on the benchmark instances given in Braun et al (2001). Factors such as task heterogeneity and resource heterogeneity are considered. Task heterogeneity depends upon the length of the jobs and size of data to be transferred for job execution. Resource heterogeneity depends on the capacity of the machine i.e. number of instructions that can be executed per second and baud rate. Both machine and task heterogeneity can have the values high and low.
PUD algorithm is analyzed with Min-min and ADA algorithm. The Min-min algorithm works as follows. It begins with the set U of all unscheduled tasks. Then, it calculates expected completion time for each of the tasks in U at all the resources. Then, the task with minimum expected completion time from unscheduled tasks is selected and assigned to the corresponding resource. Finally, the newly scheduled task is removed from U and the process repeats until all tasks are scheduled. It has a time complexity of $O(mn^2)$, where $m$ is the number of resources in the system and $n$ is the number of tasks.

In Min-min algorithm, $rt_j$ denotes the expected time in which resource $R_j$ will become ready to execute a task after finishing the execution of all tasks assigned to it. $ET_{ij}$ denotes estimated execution time of task $T_i$ on resource $R_j$. $C_{ij}$ denotes expected completion time of task $T_i$ on resource $R_j$.

```plaintext
for all tasks $T_i$ in task list $T$
for all resources $R_j$
    $C_{ij} = ET_{ij} + rt_j$
do until all tasks in $T$ are mapped
    for each task in $T$
        find the earliest completion time and the resource that obtains it
        find the task $T_k$ with the minimum earliest completion time
        assign task $T_k$ to the resource $R_j$ that gives the earliest completion time.
        delete task $T_k$ from $T$
        update $rt_j$
        update $C_{ij}$ for all $i$
end do
```

**Algorithm 4.3 Min-min algorithm**
ADA algorithm works as follows. It has two steps. The first step is determined by system performance and the other is by application demand. Initially, for all the jobs, select the resource from the set of resources on which the job can be completed with minimum time. Then a <job, resource> set is created which holds all the submitted jobs with their respective most suitable resources according to minimum completion time. However, for each resource, there may be more than one job, as per application demands and these jobs can't be assigned to their resource only according to the minimum completion time. Then, the user deadline is considered for selecting a job to be executed on a resource. By considering user deadline, jobs may be completed within their expected completion time.

The simulation is carried out in two ways. First, the number of resources is kept as 16, the number of jobs is varied from 100 to 600 and the makespan, hit rate and miss rate are measured. Second, the number of jobs is considered as 512 and the number of resources is considered as 16 to evaluate their efficiency. The efficiency of PUD algorithm is proved by comparing the results with Min-min and ADA algorithms based on makespan, hit rate and miss rate. One of the most popular optimization criteria is the minimization of the makespan. Makespan is an indicator of the productivity of a grid system. The small values of makespan mean that the scheduler is providing efficient scheduling of tasks to resources.

Makespan is defined as the time taken to finish the latest task. It is also defined as maximum of completion time of the latest job on the resources.

\[
\text{Makespan} = \max(C_{T_j}), j = 1 \text{ to } m
\]  

(4.6)

where \(C_{T_j}\) is defined as completion time of the last job at resource \(j\).
Hit rate is defined as ratio between the number of jobs completed within user deadline and the number of jobs submitted. Miss rate is inversely proportional to hit rate which is based on number of jobs that could not be completed within user deadline.

\[
\text{Hit rate} = \frac{J_{\text{comp}}}{J_{\text{sub}}} \times 100 \tag{4.7}
\]

\[
\text{Miss rate} = \frac{J_{\text{fail}}}{J_{\text{sub}}} \times 100 \tag{4.8}
\]

where

- \( J_{\text{comp}} \) – Number of jobs that are completed within user deadline
- \( J_{\text{sub}} \) – Number of jobs that are submitted
- \( J_{\text{fail}} \) – Number of jobs that could not be completed within user deadline.

Table 4.2 shows the results of various algorithms such as Min-min, ADA and PUD algorithms based on makespan for varying number of jobs. It shows that the makespan value is less in PUD algorithm than the other two algorithms.

**Table 4.2 Performance of PUD algorithm based on makespan for varied number of jobs**

<table>
<thead>
<tr>
<th>Number of Jobs</th>
<th>Makespan (Seconds)</th>
<th>Min-min</th>
<th>ADA</th>
<th>PUD</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>567</td>
<td>502</td>
<td>412</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>997</td>
<td>903</td>
<td>865</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>1567</td>
<td>1489</td>
<td>1324</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>1987</td>
<td>1768</td>
<td>1695</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>2231</td>
<td>2098</td>
<td>1989</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>2871</td>
<td>2734</td>
<td>2710</td>
<td></td>
</tr>
</tbody>
</table>
Figure 4.2 shows graph representation of makespan values of various algorithms such as Min-min, ADA and PUD algorithms. It shows that the PUD algorithm allocates jobs efficiently.

![Figure 4.2 Makespan of PUD algorithm for varied number of jobs](image)

**Figure 4.2 Makespan of PUD algorithm for varied number of jobs**

Table 4.3 shows performance analysis of algorithms based on hit rate. It is seen that the proposed PUD algorithm has maximum hit rate when compared with the other two algorithms. User satisfaction is measured based on the hit rate.

**Table 4.3 Performance of PUD algorithm based on hit rate for varied number of jobs**

<table>
<thead>
<tr>
<th>Number of Jobs</th>
<th>Hit Rate (%)</th>
<th>Min-min</th>
<th>ADA</th>
<th>PUD</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>46</td>
<td>85</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>43</td>
<td>76</td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>39</td>
<td>68</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>37</td>
<td>59</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>34</td>
<td>58</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>29</td>
<td>55</td>
<td>57</td>
<td></td>
</tr>
</tbody>
</table>
Figure 4.3 shows performance analysis of algorithms based on hit rate. The hit rate of the PUD algorithm is more when compared to the other algorithms. It shows that PUD algorithm completes more jobs within the specified user deadline.

![Figure 4.3 Hit rate of PUD algorithm for varied number of jobs](image)

To analyze the performance of various algorithms, four cases with different set of 512 jobs and 16 resources are created. Table 4.4 shows makespan values of various algorithms such as Min-min, ADA and PUD algorithms. In PUD algorithm, the makespan value is less in all the cases. Hence, PUD algorithm allocates jobs efficiently in all the cases.

<table>
<thead>
<tr>
<th>Cases</th>
<th>Makespan (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min-min</td>
</tr>
<tr>
<td>Case 1</td>
<td>2123</td>
</tr>
<tr>
<td>Case 2</td>
<td>1618</td>
</tr>
<tr>
<td>Case 3</td>
<td>2256</td>
</tr>
<tr>
<td>Case 4</td>
<td>2089</td>
</tr>
</tbody>
</table>
Figure 4.4 shows the performance of PUD algorithm based on makespan for various cases with 512 jobs and 16 resources in which the makespan of PUD algorithm is less than the other two algorithms.

Figure 4.4 Makespan of PUD algorithm for different cases

Table 4.5 shows the hit rate of Min-min, ADA and PUD algorithms. Four different cases of 512 jobs and 16 resources are taken for comparison. The PUD algorithm produces better hit rate when compared with the other two algorithms.

Table 4.5 Performance of PUD algorithm based on hit rate for different cases

<table>
<thead>
<tr>
<th>Cases</th>
<th>Hit Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min-min</td>
</tr>
<tr>
<td>Case 1</td>
<td>41</td>
</tr>
<tr>
<td>Case 2</td>
<td>38</td>
</tr>
<tr>
<td>Case 3</td>
<td>45</td>
</tr>
<tr>
<td>Case 4</td>
<td>43</td>
</tr>
</tbody>
</table>
Figure 4.5 shows the hit rate of Min-min, ADA and PUD algorithms for four different cases of 512 jobs and 16 resources. It shows that the PUD algorithm completes more jobs within user deadline. Hence, hit rate is improved.

4.4.3 UDDA Algorithm

UDDA algorithm is compared with the benchmark Min-min algorithm by considering communication time and user deadline. It is also compared with PUD algorithm by considering communication time. The simulation is done with 16 resources and the number of jobs is varied from 100 to 600.
Table 4.6 Performance of UDDA algorithm based on makespan for varied number of jobs

<table>
<thead>
<tr>
<th>Number of Jobs</th>
<th>Makespan (Seconds)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min-min</td>
<td>PUD</td>
<td>UDDA</td>
</tr>
<tr>
<td>100</td>
<td>567</td>
<td>412</td>
<td>307</td>
</tr>
<tr>
<td>200</td>
<td>997</td>
<td>865</td>
<td>840</td>
</tr>
<tr>
<td>300</td>
<td>1567</td>
<td>1324</td>
<td>1200</td>
</tr>
<tr>
<td>400</td>
<td>1987</td>
<td>1695</td>
<td>1531</td>
</tr>
<tr>
<td>500</td>
<td>2231</td>
<td>1989</td>
<td>1912</td>
</tr>
<tr>
<td>600</td>
<td>2871</td>
<td>2710</td>
<td>2456</td>
</tr>
</tbody>
</table>

Table 4.6 shows the makespan of various algorithms such as Min-min, PUD and UDDA algorithms. It is seen that the makespan value is reduced in the proposed UDDA algorithm than the other two algorithms.

In Figure 4.6, makespan is shown for Min-min algorithm, PUD algorithm and UDDA algorithm. The comparison result shows that the UDDA algorithm provides reduced makespan when compared with other algorithms such as Min-min and PUD algorithms.

Figure 4.6 Makespan of UDDA algorithm for varied number of jobs
Table 4.7 shows the performance analysis of algorithms based on hit rate. It is seen that the proposed UDDA algorithm has maximum hit rate when compared with the other two algorithms.

Table 4.7  Performance of UDDA algorithm based on hit rate for varied number of jobs

<table>
<thead>
<tr>
<th>Number of Jobs</th>
<th>Hit Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min-min</td>
</tr>
<tr>
<td>100</td>
<td>46</td>
</tr>
<tr>
<td>200</td>
<td>43</td>
</tr>
<tr>
<td>300</td>
<td>39</td>
</tr>
<tr>
<td>400</td>
<td>37</td>
</tr>
<tr>
<td>500</td>
<td>34</td>
</tr>
<tr>
<td>600</td>
<td>29</td>
</tr>
</tbody>
</table>

Figure 4.7 Hit rate of UDDA algorithm for varied number of jobs

Figure 4.7 shows the performance analysis of the algorithms based on hit rate. The hit rate of the UDDA algorithm is more when compared to the other algorithms. By considering the data transfer time of the job, the hit rate is reduced in UDDA algorithm.
Table 4.8 shows makespan values of various algorithms such as Min-min, PUD and UDDA algorithms. In UDDA algorithm, the makespan value is reduced by considering the data transfer time while assigning jobs to the resources.

Table 4.8  Performance of UDDA algorithm based on makespan for different cases

<table>
<thead>
<tr>
<th>Cases</th>
<th>Makespan (Seconds)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min-min</td>
<td>PUD</td>
<td>UDDA</td>
</tr>
<tr>
<td>Case 1</td>
<td>2123</td>
<td>2054</td>
<td>1920</td>
</tr>
<tr>
<td>Case 2</td>
<td>1618</td>
<td>1610</td>
<td>1513</td>
</tr>
<tr>
<td>Case 3</td>
<td>2256</td>
<td>2201</td>
<td>1997</td>
</tr>
<tr>
<td>Case 4</td>
<td>2089</td>
<td>2003</td>
<td>1901</td>
</tr>
</tbody>
</table>

Figure 4.8 shows the performance of UDDA based on makespan for various cases with 512 jobs and 16 resources in which the makespan is less for UDDA algorithm than the other two algorithms.

Figure 4.8  Makespan of UDDA algorithm for different cases
Table 4.9 shows the hit rate of Min-min, PUD and UDDA algorithms. For four different cases, UDDA algorithm produces better hit rate when compared with the other two algorithms.

Table 4.9  Performance of UDDA algorithm based on hit rate for different cases

<table>
<thead>
<tr>
<th>Cases</th>
<th>Hit Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min-min</td>
</tr>
<tr>
<td>Case 1</td>
<td>41</td>
</tr>
<tr>
<td>Case 2</td>
<td>38</td>
</tr>
<tr>
<td>Case 3</td>
<td>45</td>
</tr>
<tr>
<td>Case 4</td>
<td>43</td>
</tr>
</tbody>
</table>

Figure 4.9 shows the hit rate of Min-min, PUD and UDDA algorithms. The UDDA algorithm produces a better hit rate when compared with the other two algorithms.

Figure 4.9  Hit rate of UDDA algorithm for different cases
4.5 CONCLUSION

The analysis of experimental results shows that PUD algorithm is better with reduced makespan than Min-min and ADA algorithms. PUD algorithm has reduced makespan on various heterogeneous environments such as task heterogeneity and machine heterogeneity. PUD algorithm also has a better hit rate which shows that more number of jobs are completed within user deadline.

In UDDA algorithm, user satisfaction is improved by considering data transfer time of the jobs. It is observed that the UDDA proves to be an efficient scheduling algorithm when compared with the existing algorithms in terms of makespan, hit rate and miss rate. The next chapter proposes a grouping based user demand aware job scheduling algorithm for effective utilization of resources in grid environment. Since the resources in the grid environment are dynamic in nature, the availability time of the resources needs to be considered for effective utilization of the resources.