CHAPTER 5

A NOVEL DYNAMIC RESOURCE ALLOCATION SCHEME FOR COMPUTATIONALLY INTENSIVE APPLICATIONS

Grid computing is a form of parallel and distributed computing and suitable for providing cost effective solution for special type of both computationally and data intensive problems such as sequence alignment, weather forecasting, etc. Solving this type of application problems in grid needs a judicious allocation scheme to solve them awfully fast. Most of the grid tasks are scheduled based on the First Come First Served (FCFS) or FCFS with advanced reservation, Shortest Job First (SJF) etc. But these traditional algorithms seize more computational time due to soaring waiting time of jobs in job queue before being allocated to suitable resources. In grid resource allocation, the resources discovery and selection for a waiting job is an NP-complete problem. To triumph over this problem, there is a need for an effective dynamic allocation scheme to provide an approximate solution.

5.1 INTRODUCTION

A computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities. According to its function, Grid is classified into three types: Computing Grid, Data Grid, and Service Grid. Computing Grid is used to connect varied computing resource on the network to construct a virtual high performance computer, which could offer high performance computer (Ranganathan and Foster 2002). The traditional computing Grid systems involve many technologies such as certification, task scheduling,
communication protocols, fault tolerance and so on. The task of Grid resource broker and scheduler is to dynamically identify and characterize the available resources, to select and allocate the most appropriate resources for a given waiting job (Mitrani and Palmer 2003). The resources are typically heterogeneous and locally administered, accessible under different local policies. Advance reservation (Foster et al 2004) is currently being added to Portable Batch System (PBS).

In a Grid resource allocation, the mapping of grid resources and an independent job in optimized manner is so hard where optimized mapping could not be predicted. This thesis proposed a combination of uninformed search and informed search method using heuristic algorithm to provide the good optimal solution by mapping the appropriate resources to jobs and takes minimal allocation time that will consequently minimize the average waiting time of the jobs in the queue. A heuristic algorithm is an algorithm that directs the search in the most profitable direction and it will not guarantee that it will never lead to wrong direction. In other words, it ignores whether the solution to the problem can be proven to be correct, but which usually produces a good solution. Heuristics are typically used when there is no known way to find an optimal solution, or when it is desirable to give up finding the optimal solution for an improvement in run time.

Here, our primary objective is to investigate effective resource allocation techniques on the basis of computational economy through simulation. We considered simulating millions of resources and thousands of users with varied requirements and study scalability of systems, algorithms, efficiency of resource allocation policies and satisfaction of users. In our simulation, we have modeled an application in the area of biotechnology and this can be extended to astrophysics, network design, and high-energy physics in order to study the usefulness of our resource allocation techniques. The
results of our work will have significant impact on the way resource allocation is performed for solving the special type of applications which is mentioned at the beginning of this chapter using Grid computing systems.

5.2 RELATED WORKS

Job scheduling in parallel system has been extensively researched in the past. Typically, this research has focused on allocating a single resource type (e.g., CPU usage) to jobs in the ready queue. The use of many of these scheduling algorithms has been limited due to restriction in application designs, runtime system, or the job management system itself. Therefore, simple allocation scheme such as First Come First Served (FCFS) or FCFS with First fit back Fill (FCFS/FF) are used in practice (Vijay Subramanian et al 2002).

Current job scheduling practices typically support variable resource allocation to a job, and run to completion scheduling. Scheduling policies are also heavily based on First Come First Served (FCFS) methods (Bitten et al 2000). A FCFS scheduling algorithm allocates resources to jobs in the order that they arrive. The FCFS algorithm schedules the next job in ready queue as soon as sufficient system resources become available to meet all the job requirements. The advantage is that this provides a level of determinism on the waiting time of each job (Ememann et al 2002). The disadvantage of FCFS shows up when the jobs at the head of the ready queue cannot be scheduled immediately due to insufficient system resources, but, jobs further down the queue would be able to execute the given currently available system resources. These latter jobs are essentially blocked from execution while the system resource remains idle.

Stefka Fidanova (2006) compared the simulated annealing approach with the ant algorithm for scheduling jobs in Grid. Marek Mika et al
(2004) formulated the scheduling problem as a linear programming problem and proposed local search meta-heuristic to schedule workflow jobs on a Grid. Fair share scheduling (Doulamis et al 2007) is compared with simple fare task order scheduling, adjusted fair task order scheduling and max-min fair share scheduling algorithm is developed and tested with the existing scheduling algorithms.

Moreno and Alonso-Conde (2003) address the issues that the resource broker has to tackle like resource discovery and selection, job scheduling, job monitoring and migration etc. Resource Management System (RMS) (Yuan-Shun Dai et al 2008, Yuan-Shun Dai and Gregory Levitin 2007) was discussed and models of grid RMS availability by considering both the failures of Resource Management (RM) Servers and the length limitation of request queues. Resource management systems can divide service tasks into Execution Blocks (EB), and send these blocks to different resources. To provide a desired level of service reliability, the RMS assigns the same EB to several independent resources for parallel execution.

5.3 CONCEPTUAL MODEL OF SWIFT SCHEDULER (SS)

The Swift Scheduler allocates resources to data interdependent parallel task for solving the problems like sequence alignment. The best-first algorithm can be used to discover suitable resource for a waiting task with reference to its parent or data dependent task.

Let $N$ be the number of jobs in job queue ‘$J$’ which is indicated as,

$$J = \{J_1, J_2, J_3, \ldots, J_N\}$$  \hspace{1cm} (5.1)

Jobs are allotted to $M$ number of resources in resource queue ‘$R$’ which is indicated as,
Let $F(J_i, R_j)$ be the cost (which will be calculated according to the requirement of the application problem such as overall job completion time, memory usage and network bandwidth usage of job, etc) for the $i^{th}$ job in $j^{th}$ resources be calculated as,

$$F(J_i, R_j) = G(J_i, R_j) + H(J_i, R_j) \quad (5.3)$$

Let $G(J_i, R_j)$ be the original path cost (which includes the combination of original job completion time, memory and network requirements of jobs) of the $i^{th}$ job to the reference resource of $j^{th}$ resources which can be calculated as,

$$G(J_i, R_j) = \sum_{i=0}^{N} \sum_{j=0}^{M} \left( J_{Li} / RC_j \right) \quad (5.4)$$

where $J_{Li}$ be the job length of $i^{th}$ jobs and $RC_j$ be the capacity of the $j^{th}$ resources.

Let $H(J_i, R_j)$ be the admissible heuristic function or expected path cost of the $i^{th}$ job in $j^{th}$ resources which can be calculated as

$$H(J_i, R_j) = \sum_{i=0}^{N} \sum_{j=0}^{M} \left( J_{Li} / RC_j \right) + Network_{latency} \quad (5.5)$$
5.4 PROPOSED ARCHITECTURE AND ALGORITHM OF SWIFT SCHEDULER (SS)

(a) Architecture of swift scheduler

The following Figure 5.1 shows the architecture of our proposed dynamic resource allocation algorithm called Swift Scheduler which comprises three main modules namely Job Manager, Resource Manager and Resource Allocation Mechanism.

![Figure 5.1 Architecture of Swift Scheduler](image)

The role of job manager is to collect all incoming jobs from parallel application and group them according to their interdependence. These group jobs are maintained in a job list. The group jobs in Job lists are moved to job ready queue in First Come First Served basis when it is ready for execution. The resource manager collects all available resources which are distributed over the network and which maintain resource’s characteristics in Global
Resource Index Information Table (GRIIT). Each site maintains its characteristics as well as its neighbors in Local Resource Information Table (LRIT). The resource allocation mechanism in SS has two main services namely resource discovery services and scheduling.

The function of resource discovery service is to collect the jobs from job ready queue and search the best optimal resources from the GRIIT based on jobs requirements (i.e., computational power, memory requirement and network quality) by using A* heuristic function. Once the resource discovery service has found the best optimal resource for particular job, the scheduler allocates the particular jobs to particular resources.

(b) Implementation of proposed dynamic resource allocation algorithm

The proposed novel dynamic resource allocation mechanism using Swift Scheduling allocates the particular jobs to best optimal resources based on their requirements and resource capability using A* heuristic search function. All the incoming jobs to job manager from different grid users are maintained in unordered job list and are transferred to job ready queue in First Come First Serve manner when job gets ready state. If a job doesn’t get the best optimal resource, then it will be allocated to a near optimal resource.

The jobs in job ready queue send the jobs to Resource Allocation Module in First Come First Serve manner. Once the job is received by the Resource Allocation Module, it will forward the jobs to resource discovery service for discovering the best optimal resource. The resources and their characteristics are maintained in two different tables which are used in computational grid and our solution is based on a Distributed Hash Table (DHT). A DHT is a distributed and often a decentralized mechanism for associating hash values (keys) with some kind of content. Participants in the
DHT each store a small section of the contents of the hash table. The main advantage of DHTs is their scalability: a typical search on a DHT requires only $O(\log(n))$ network traffic where $n$ is the number of participants in the DHT.

(i) Global Resource Index Information Table (GRIIT): In resource manager module, it stores resource list that can be offered.

(ii) Local Resource Information Table (LRIT): It stores its resources and the resources of its neighbors.

GRIIT and LRIT have the same content that consists of seven entries as given below:

1. Provider IP: Physical address is used to discern the resource provider.
2. Usable Memory_MB: The quantity of memory that can be used by the resource provider.
3. CPU_MHz: The grade of CPU that can execute the task.
4. Storage_GB: The capacity of storage that can be used for task.
5. Network_Kbps: The resource provider periodic ping RM and derive out the quality of network according to round trip time.

GRIIT and LRIT are triggered update. This will avoid the traffic which is transferred over the network for updating LRIT and GRIIT. Once the job is received by the resource discovery service in resource allocation module, it will check the CPU, memory and network demands of jobs and finds the least cost path from submission node to execution/computing node. It uses a distance-plus-cost heuristic function (denoted as $f(x)$) to determine
the order in which the search visits nodes in the tree. The distance-plus-cost heuristic is a sum of two functions: the path-cost function (denoted as $g(x)$, which is determined by the cost of the reference resource) and an admissible "heuristic estimate" of the distance to the goal (denoted as $h(x)$). The path-cost function $g(x)$ is the cost from the submission node to the execution/computing node.

(c) **Pseudo code for Swift Scheduler**

```plaintext
procedure Swift Scheduler()
begin:
initialize Job List and Resource List;
Loop exec for i =1 to ‘N’ Jobs
begin:
    Get job’s execution requirements such as memory need, CPU speed, etc and available resources with their characteristics from resource manager as input;
    initialize path cost $f(x)$, expected path cost $h(x)$ and original path cost $g(x)$;
    if (Job’s status = = ‘Ready’)
    begin:
        transfer jobs to Job Ready Queue;
    end;
    Loop exec for j= 1 to ‘M’ resources
begin:
    calculate the pathcost of jobs by extracting the paths from submission node to M resources based on Jobs requirements;
end;
Apply A* algorithm to discover the suitable resource
```
If the discovered resource matches the job requirement, then Assign the \( i^{th} \) jobs to \( j^{th} \) resource; else

Find the nearest optimal resource and allocate the \( i^{th} \) Job;

end;

end;

5.5 PERFORMANCE EVALUATION

5.5.1 Experimental Setup

The scheduling paradigm followed in this thesis is that of offline or batch mode scheduling of a set of independent tasks (Maheswaran et al 1999). The general problem on creating a schedule for a set of jobs to run on distributed resources is called list scheduling and is considered to be NP-complete (Braun et al 2001). Many approximate heuristics have been devised for this problem and a short survey of them has been presented by (Braun et al 2001).

The resource broker is able to identify the resources that meet minimum requirements of the application such as architecture (instruction set), operating system, storage threshold and data access permissions and present them as suitable candidates for job execution to the scheduler. There are two parts in a scheduling strategy: mapping and dispatching. The jobs have to be matched to a set of resources and ordered depending on the objective function (mapping) and then sent to remote resources for execution (dispatching).
The simulation environment is constructed to evaluate our dynamic resource allocation algorithm called Swift Scheduler for heterogeneous resources and job environment. The GridSim toolkit simulator is used to simulate the SS algorithm and compare its performance with the existing algorithms. Each task has a resource requirement which is measured in Millions of Instructions Per Second (MIPS). The available processing resources or execution rate of each resource is measured in Millions of Floating Operations (MFLOPs) per second.

Up to 1500 heterogeneous tasks are scheduled onto 500 heterogeneous resources. For these experiments, each resource is assumed to have a fixed execution rate, measured in Mflop/s and jobs are characterized with MIPS. The aim of these experiments is to show that predicting the communication costs in advance will improve the efficiency. All schedulers are presented with the same set of tasks for scheduling and the same information is available to all.

5.5.2 Results and Observations

The performance of dynamic resource allocation algorithm Swift Scheduler is compared with the existing Simple Fair Task Order Scheduling (SFTO) against a large set of independent jobs with varying size and a large number of heterogeneous resources. It is assumed that the arrival rates of jobs are based on the Poisson distribution. The following Figures 5.2 and 5.3 show the job allocation methods used in SFTO and SS respectively and Tables 5.1 and 5.2 show the arrival order of the particular jobs, the time when the job will start its execution in the particular resource and its service time of the jobs in the particular resource where the selection of resources are based on the algorithms.
For example, in SFTO, the jobs J0 and J3 are allotted in resource R1. The residing time ($T_r$) of jobs J0 is the combination of jobs J0 waiting time ($T_w$) in queue and service time ($T_s$) of J0 (i.e. $T_r = T_w + T_s = 2820.03\, \text{ms} + 61.35\, \text{ms} = 2881.38\, \text{ms}$). Similarly, the residing time of job J3 is 1490ms. Job J2 is allotted in resource R2 where the residing time of J2 is 1257.19ms and jobs J4 and J1 are allotted in resource R3 where the residing time of J4 is 1886.86ms and J1 is 3391.73ms. The average waiting time of all the jobs in grid system is 2181.56ms.

**Table 5.1 Job, Resource, Start and Residing time for SFTO**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Job ID</th>
<th>Start time in ms</th>
<th>Residing time in ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>3</td>
<td>1330.66</td>
<td>1490.66</td>
</tr>
<tr>
<td>R1</td>
<td>0</td>
<td>2820.03</td>
<td>2881.38</td>
</tr>
<tr>
<td>R2</td>
<td>2</td>
<td>1214.79</td>
<td>1257.19</td>
</tr>
<tr>
<td>R3</td>
<td>4</td>
<td>1851.12</td>
<td>1886.86</td>
</tr>
<tr>
<td>R3</td>
<td>1</td>
<td>3337.99</td>
<td>3391.73</td>
</tr>
</tbody>
</table>

**Figure 5.2 Job and Resource Allocation for SFTO**
In SS, resource selection and jobs allocation are based on heuristic searching algorithm on SJF, which reduces the average waiting time of the jobs in queue. So, overall turn around time is reduced and resource utilization is increased. For example, in SS, the jobs J2 and J0 are allotted to resource R1 where its residing time is 1316.02 ms and 2980.39 ms respectively. Similarly, Jobs J3, J1 and J4 are chosen by resources R2, R2 and R3 respectively where the residing time of Jobs J3, J1 and J4 are 1178.29 ms, 2772.17 ms and 1489.25 ms respectively. The average waiting time of all jobs in grid system is 1947.22 ms which is less than SFTO. The statistical data presented here is acquired by averaging the scheduling performance over different runs.

**Table 5.2 Job, Resource, Start and Residing time for SS**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Job ID</th>
<th>Start time in ms</th>
<th>Residing time in ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>2</td>
<td>1210.02</td>
<td>1316.02</td>
</tr>
<tr>
<td>R1</td>
<td>0</td>
<td>2805.39</td>
<td>2980.39</td>
</tr>
<tr>
<td>R2</td>
<td>3</td>
<td>1093.04</td>
<td>1178.29</td>
</tr>
<tr>
<td>R2</td>
<td>1</td>
<td>2688.17</td>
<td>2772.17</td>
</tr>
<tr>
<td>R3</td>
<td>4</td>
<td>1453.51</td>
<td>1489.25</td>
</tr>
</tbody>
</table>

**Figure 5.3 Job and Resource Allocation for SS**
Our proposed algorithm Swift Scheduler (SS) is compared with FCFS, SJF, Simple Fare Task Order (SFTO) (Doulamis et al 2007) and Max-Min Fair Share (MMFS) scheduler (Doulamis et al 2007). SS is evaluated in GridSim by varying the number of resources, number of jobs against total processing time, cost, and resource utilization. The resources are varied from 100 to 500. For experimental study purpose, sample simulation results are shown in Figures 5.4 to 5.15. By analyzing the obtained results from the simulator, it is clear that the dynamic resource allocation algorithm Swift Scheduler (SS) has completed all jobs with minimum time by utilizing maximum amount of resources.

(a) Total processing time analysis

The total processing time is defined as the time it takes to complete a prescribed job or task which includes the transit time, communication time, waiting time in queue and execution time. The overall output is increased by completing the job with minimal time. The total processing time analysis shown in Figures 5.4 to 5.7, measured the total processing time by varying the jobs and resources and by fixing the resources and jobs as constant. From analyzing the graph, the jobs are increased from 250 to 1500 and resources are set as 200 and 400. The total processing time is reduced for SS while compared to other scheduling algorithms due to optimal selection of resources and efficient allocation by the scheduler.
Figure 5.4  Number of Jobs Vs Total Processing Time – Number of resources=200

Figure 5.5  Number of Jobs Vs Total Processing Time – Number of resources=400
Figure 5.6  Number of Resources Vs Total Processing Time–Number of Jobs=750

Figure 5.7  Number of Resources Vs Total Processing Time–Number of Jobs=1500
Figures 5.6 and 5.7 show the performance analysis of the total processing time for SS. The resources are varied from 100 to 500 and jobs are fixed as 750 and 1500. When the resources are increased, the total processing time is reduced, because the jobs get more resources and are able to select optimal resources for the job and complete it fast. The dynamic resource allocation scheduler SS efficiently allocate the job and resources, so 10 to 15 percentage of the total processing time is reduced compared to other schedulers.

(b) Total cost analysis

The resource cost analysis determines the cost of a resource during a specific time period to run the job on that particular resource. The costs associated with a particular resource can vary depending on the time period when the resource is used. The resource cost analysis is used to compare the costs of different resources, which enables to set resource requirements using the resource that has the lowest cost. The cost analyses are shown in the Figures 5.8 to 5.11. These analyses are carried out between the number of jobs versus total cost and number of resources versus total cost by setting the resource and jobs as constant.
Figures 5.8 and 5.9 show the resource utilization total costs analysis for fixed number of resources. When increasing the number of jobs from 250 to 1500 for fixed number of resources, the total cost is increased. When the processing time of job is increased, the total cost is also increased. But the
proposed SS completes all the jobs with minimum cost compared to other resource allocation mechanisms.

Figure 5.10 Number of Resources Vs Total Cost – Number of Jobs=750

Figure 5.11 Number of Resources Vs Total Cost – Number of Jobs=1500
Figures 5.10 and 5.11 show the cost analysis of SS by varying the resources and number of jobs as 750 and 1500. When the resources are increased from 100 to 500 and jobs are set as 750 and 1500, the total cost is decreased. The numbers of resources are increased for same set of jobs, which are running in different resources. So, the total processing time and the cost also are reduced. The proposed dynamic resource allocation scheduler SS efficiently allocates the jobs and resources and outperforms well while compared to other schedulers and reduces the resource utilization cost.

(c) **Resource utilization analysis**

Grid computing is a cooperative architecture and a decentralized model where each resource is capable of helping scheduling and can offload locally scheduled jobs to other cooperating resources. This enables dynamic workload sharing to maximize hardware resource utilization and high availability to ensure service delivery for the grid users. The following Figures 5.12 to 5.15 show the resource utilization analysis by varying the numbers of jobs and number of resources by setting the resource and jobs as constant.
Figure 5.12 Number of jobs Vs Resource Utilization – Number of Resources=200

Figure 5.13 Number of jobs Vs Resource Utilization – Number of Resources=400

Figures 5.12 and 5.13 show the resources utilization analysis graphs which are plotted against the number of jobs versus resource utilization.
setting the resource as 200 and 400. When the job size is increased, resource utilization is also increased. The proposed SS utilizes the grid resources highly while compared to other scheduling algorithms.

![Bar Graph](image1)

**Figure 5.14** Number of Resources Vs Resource Utilization – Number of Jobs=750

![Bar Graph](image2)

**Figure 5.15** Number of Resources Vs Resource Utilization–Number of Jobs=1500
Figures 5.14 and 5.15 are plotted against the number of resources versus resource utilization setting the number of jobs as 750 and 1500. When the resources are increased from 100 to 500, the batch of jobs is run on the varying resources. The resources are increased for the fixed number of jobs, the resource utilization is reduced, because more resources are available and distributed to all resources. So, the overall resource utilization is reduced. The proposed dynamic resource allocation scheduler SS highly utilizes the grid resources compared to other scheduling algorithms.

5.6 SUMMARY OF CONTRIBUTION

The dynamic resource allocation mechanism uses the proposed Swift Scheduler which completes a task by utilizing low cost resources with minimum computational time. The proposed SS uses the heuristic function to select the best available resources to achieve a higher throughput while maintaining the desired success rate of job completion. However, in all conditions, the proposed algorithms outperform the other resource allocation schedulers. In SS, the job and resource allocation policy is more effective than FCFS, SJF, SFTO and MMFS, in the extent of computational complexity with lower cost but higher resources utilization. The proposed SS uses the heuristic search algorithm to select the optimal resources which combine with traditional Shortest Job First (SJF) algorithm for mapping the jobs. So, it reduces the total processing time and cost and maximizes the resource utilization.