2005

Genetic algorithm based scheduler for computational grids.

Mona Aggarwal
University of Windsor

GENETIC ALGORITHM BASED SCHEDULER FOR COMPUTATIONAL GRIDS

by

Mona Aggarwal

A Thesis
Submitted to the Faculty of Graduate Studies and Research
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada
2004

Your file Votre référence
ISBN: 0-494-04978-2
Our file Notre référence
ISBN: 0-494-04978-2

Abstract

Computational grids can be used to solve grand challenge problems of scientific research and to handle peak processing demands in large organizations. For managing these tasks, grids may use distributed and heterogeneous compute resources, including commodity machines and supercomputers.

This thesis deals with highly scalable distributed resource management architecture for the global grid. The main component in the architecture is the grid scheduler. A scheduler must use the available resources efficiently, while satisfying competing and mutually conflicting goals. Genetic Algorithm is used to obtain the best schedule for mapping of tasks to compute-nodes. The grid workload may consist of multiple jobs, with quality-of-service constraints. Each job has tasks with arbitrary precedence constraints and arbitrary processing time. A Directed Acyclic Graph (DAG) represents each such job.

The thesis presents the design, implementation and test results for a genetic based grid scheduler. It attempts to minimize make-span, idle time of the available computational resources, turn-around time and the specified deadlines by the users. The architecture is hierarchical and the scheduler is usable at either the lowest or the higher tiers. It can also be used in both the intra-grid of a large organization and in a research grid consisting of large clusters, connected through a high bandwidth dedicated network.

Dedication

This work is dedicated to my parents.

Acknowledgments

I am highly indebted to my thesis advisors Dr. Robert Kent and Dr. Ngom, who have deep knowledge and great patience to discuss the ideas with me. They offered an environment in which I could work on creative ideas. It has been a privilege to be their student.

I would like to express my special thanks to my committee members Dr. Kemal Tepe and Dr. Arunita Jaekel for their valuable suggestions and guidance. I am also highly thankful to Dr. Ezeife Christie for agreeing to chair the committee.

I am highly thankful to my friends Gopal, Neeta, Badar and members of our research group for their unstinted support.

I would like to take this opportunity to thank my parents, my brother Aditya and my sister-in-law Seema for giving me inspiration and unconditional love and support throughout my career.

Finally, I must greatly appreciate my husband Nilesh for his patience and support for my higher studies.

Table of Contents

Abstract iii

Dedication iv

Acknowledgements v

List of Tables viii

List of Figures ix

1 Computational Grids 1

1.1 Introduction 1
1.2 Grid Scheduling 2
1.3 Resource Management System (RMS) in Grids 3
1.3.1 The Resource Information Service 5
1.4 Globus Architecture for Grid Scheduling 5
1.5 Globus Tool-kit: An Emerging Standard 7
1.6 Schedulers for the Global Grid 8
1.7 Problem Statement 8
1.8 Contributions 8
1.9 Organization of the Document 9

2 Current Schedulers for Computational Grids 10

2.1 Introduction 10
2.2 Evolution of Grid Computing and Grid Schedulers 11
2.3 Cluster Scheduling 12
2.4 Grid Scheduling Approaches 13
2.5 Existing Grid Schedulers 14
2.6 Condor 16
2.7 Condor-G 17
2.8 AppLeS Scheduler 18
2.9 Nimrod-G Resource Broker 20
2.10 GrADS Scheduler 21
2.11 Genetic Algorithm for Grid Schedulers 22
2.11.1 TITAN GA-based Scheduler 22
2.12 Comparison of Schedulers 25
2.13 Conclusion 26

3 A new GA Based Scheduler 27

3.1 Introduction 27
3.2 The Hierarchical Architecture of Resource Allocation Systems 28

3.3 A Resource Allocation System 29
3.4 The Directed Acyclic Graph (DAG) Model for a Job 31
3.5 A Genetic Algorithm based Scheduler 33
3.6 The Genetic Algorithm 33
3.7 The Chromosome 34
3.8 Fitness Function 35
3.9 The First Component: Turn-around time 36
3.10 The Second Component: Deadline time 36
3.11 The Third Component: Makespan 37
3.12 The Fourth Component: Node Gap Time 37
3.13 The Fitness Function 38
3.14 Generation of a New Population 38
3.15 Crossover Process 39
3.16 Mutation Process 39
3.17 New Chromosome through Elitism 40
3.18 The GA-based Grid Scheduling Algorithm 41
3.19 Conclusion 41

4 Experimental Results and Discussions 42

4.1 Introduction 42
4.2 The Complete Simulation Process 43
4.3 Test Environment for Genetic Algorithm-based Grid Scheduler 44
4.4 Tests with Standard DAG jobs 45
4.5 Tuning the Parameters of the Genetic Algorithm 47
4.6 Comparison with NIMROD/G scheduler through Gridsim 52
4.7 Conclusion 54

5 Conclusion and Future Work 55

Bibliography 57
Grid System Taxonomy 61

Vita Auctoris 63

List of Tables

Table 1 Comparison of Schedulers Characteristics 26
Table 2: Measured Makespan vs Optimum Value for STG Jobs with 50 to 3000 Tasks 46
Table 3 Makespan Values Obtained through Nimrod-G-Gridsim Simulator and GA based Scheduler 53

List of Figures

Figure 1-1 General Framework of a Computational Grid [Buyya, 2000b, pp: 1] 2
Figure 1-2 Scheduling systems[Casavant, 1988, pp: 142] 3
Figure 1-3 Resource Management System Abstract Structure (Buyya, 2002a, pp: 5) 4
Figure 1-4 GARA Resource Management Architecture (Czajkowski, 1997, pp: 12) 6
Figure 1-5 Major Components of GRAM Implementation (Foster, 1999, pp: 4) 7
Figure 2-1 Architecture of a condor pool with no jobs running (Wright, 2001, pp: 4) 16
Figure 2-2 Remote execution by Condor-G on Globus-managed resources (Frey, 2002, pp: 4) 18
Figure 2-3 Steps in the AppLeS methodology (Berman, 2003, page: 2) 19
Figure 2-4 Nimrod/G and Globus Components Interactions (Buyya a, 2000, pp: 4) 20
Figure 2-5 GrADS Program Preparation and Execution Architecture (Berman, 2001, pp: 8) 21
Figure 2-6 The TITAN architecture (Spooner, 2003a, pp: 3) 23
Figure 2-7 Chromosome Representation (Spooner, 2002b, pp:8) 24
Figure 2-8 Crossover Operation (Spooner, 2002b, pp: 8) 24
Figure 2-9 Mutation Operation (Spooner, 2002b, pp: 8) 25
Figure 3-1 A Hierarchical Architecture of Resource Allocation System (s) 29
Figure 3-2 A Resource Allocation System 30
Figure 3-3 A DAG Model for a Job 31
Figure 3-4 The Job File for a Standard Task Graph (STG) - (DAG model) 32
Figure 3-5 Genetic Algorithm 34
Figure 3-6 Representation of a Chromosome 34
Figure 3-7 Sample of a Population of Chromosomes 34
Figure 3-8 Crossover Process 39
Figure 3-9 Mutation Process 40
Figure 4-1 Process Flow of Simulation 43
Figure 4-2 Graphical User Interface 44
Figure 4-3 The measured Makespan versus the Optimum Values 47
Figure 4-4 Fitness versus the Crossover Ratio 48
Figure 4-5 Fitness versus # of Generations 49
Figure 4-6 Standard Deviation versus # of Generations 50
Figure 4-7 Fitness versus # of Generations for 10 Jobs of 3650 Tasks 51
Figure 4-8 Fitness versus # of Generations for 35 Jobs of 10750 Tasks 51
Figure 4-9 Comparison of Nimrod-G and GA – based Scheduler 54

Chapter 1

1 Computational Grids

1.1 Introduction

"A computational grid is a hardware and software infrastructure that provides dependable consistent, pervasive and inexpensive access to high-end computational capabilities" [Foster, 1998].

Today the users visualize a grid in a number of different ways. A number of academic-research organizations have connected the clusters at Universities, through high-speed dedicated networks, for high performance computing as well as for grid research. In general a cluster consists of homogeneous, dedicated and tightly coupled workstations that may be used to process high-performance jobs. A grid, on the other hand, consists of heterogeneous and intermittently available workstations. Whereas a cluster is expected to be located at a single laboratory, the workstations on a grid are widely distributed geographically. Moreover a grid is expected to be used for a wide variety of applications. The resources on the grid may include many data repositories, code repositories and compute-nodes.

A scheduler system will be the interface between a user and the grid resources. Scheduling of jobs on a grid or a cluster is the task of mapping jobs to the available compute-nodes. On a grid, which is dynamic, heterogeneous and geographically spread out and which caters to highly diverse applications, scheduling is a NP-complete problem [Buyya, 2000b].

Grids have evolved from a natural progression of parallel processing systems and distributed processing systems.

Parallel processing systems can be classified into two types:

- Shared Memory Processing systems
- Clusters of workstations connected through an interconnection network.

Centralized systems designed to cater to the needs of a large number of users are usually multi-processor systems with a shared memory. A distributed system consists of heterogeneous

systems connected in parallel. A grid is a dynamic and geographically spread-out distributed system.

Figure 1-1 shows the general framework of a computational grid with Domain Resource Managers (DRM) located at different places in the world. Similarly the clients may also belong to any place on the globe. Buyya et al [Buyya, 200b] call the Resource Management System as the Grid Resource Broker.

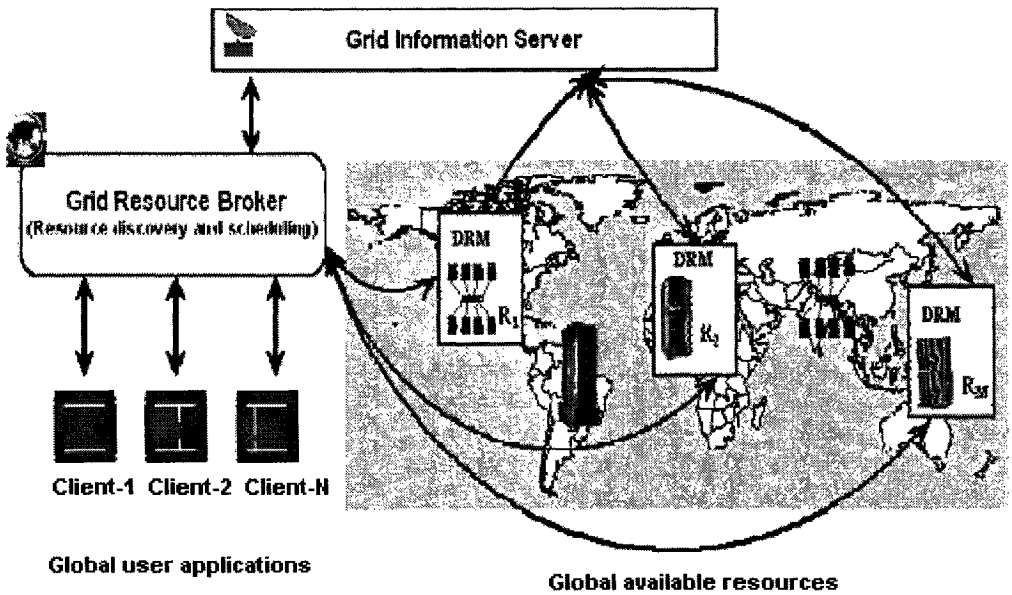

Figure 1-1 General Framework of a Computational Grid [Buyya, 2000b, pp: 1]

1.2 Grid Scheduling

Grid scheduling is used to "efficiently and effectively manage the access to grid resources by the users" [Buyya, 2002a]. According to [Casavant, 1988], the scheduling problem consists of three main components.

- Consumer(s).
- Resources(s).
- Policy.

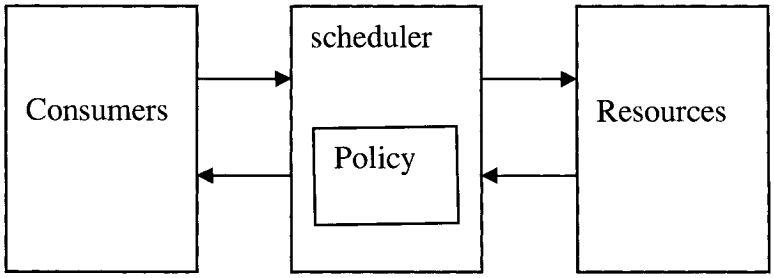

Figure 1-2 Scheduling systems [Casavant, 1988, pp: 142]

The policy may specify the objectives that a scheduling system may satisfy. It may also specify, at the implementation level, the method of mapping jobs to resources. Example of such a policy may be First Come First Serve (FCFS), Shortest Job First and Deadline Sort etc. The policy chosen for implementing a scheduler would affect both users (called consumers by Casavant) and the resource providers.

There are some common objectives that a scheduler has to satisfy for a parallel, distributed or for grid system. According to [Wright, 2001], a scheduler is designed to satisfy one or more of the following goals:

- Maximizing system throughput.
- Maximizing resource utilizations.
- Maximizing economic gains.
- Minimizing the turn-around time for an application.

Scheduling has to take into account the objectives of the service provider, the needs of the user and the requirements of the applications. In addition a scheduler can also be designed to follow a specific or an adaptable scheduling policy. Optimization of scheduling process can be attempted when performance goals and scheduling policies have been defined.

1.3 Resource Management System (RMS) in Grids

A scheduler is a component of the resource management system on a grid. Figure 1-3 shows the architecture of a general RMS. Resource Information from the sensors of the service-provider forms one input of RMS. The other input is the Resource Requests from the users.

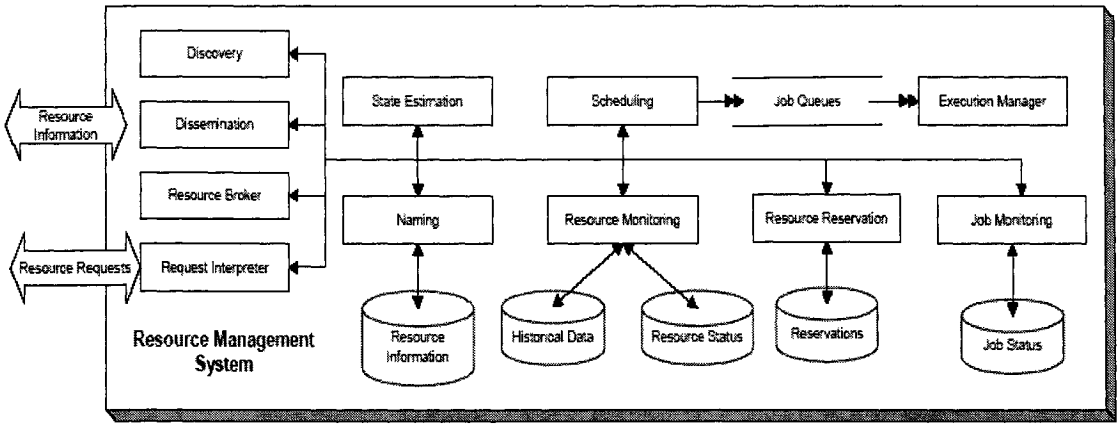

Figure 1-3 Resource Management System Abstract Structure (Buyya, 2002a, pp: 5)

The resource information inter-acts with the *Discovery* module of RMS. The *State Estimation* system converts the information, through the *Naming* system to a *Resource Information* database. The *Resource Monitoring* system maintains the *Historical Data* and the present *Resource Status* databases. The *Dissemination* system may be used to convey the resource information to users on the grid. The *Request Interpreter* may format the request, or in some cases like the TITAN system [Spooner, 2003a], it may generate estimates of execution time for each task of the job request. The *Resource Broker* will negotiate for the resources. It may use *Resource Reservation* module to save information about the reserved resources in the *Reservations* database. The *Scheduling* system will map the jobs to resources and add them in the *Job Queues*. Then the *Execution Manager* sends the job for execution to the allocated resources. The *Job Monitor* continuously monitors the job, as it is being executed. It stores the state of the job at checkpoints in the *Job Status* database.

[Foster, 1998] defines the responsibilities of a resource management system on the grid as the "discovery of available resources for an application, mapping the resources to the application subject to some performance goals and scheduling policies and loading the application to the resource in accordance with the best available schedule."

In addition a resource management system must provide the facility of checkpointing so that a job can be migrated, along with its state, to a different node. Migration may be required in case a pre-emptive scheduling policy is considered due to failure of some nodes or some part of the network. Facilities of suspension of processing, resuming the processing or aborting a job are also usually provided in such systems.

1.3.1 The Resource Information Service

The resource information service in a grid may be centralized, decentralized or hybrid. A centralized service contains information about the whole of the grid. Thus theoretically it may be able to lead to optimized scheduling. A basic disadvantage at the centralized information service is a single point of failure. An example of a centralized information service is the Globus Meta Directory Service (MDS) given by Foster. [Foster, 1997a].

The Decentralized information service will store information about a node at the node itself. Querying the node can provide the necessary information to a scheduler. But the querying process may have a large overhead. Network Weather Service [Wolski, 2003a] is an example of a decentralized resource information service.

The hybrid scheme is a hierarchical scheme where at the highest level of a group, the information is stored in a centralized entity. Centralized entity contains information about the resources of the entire system. But the information about each resource or a group of resources is stored in a decentralized manner.

1.4 Globus Architecture for Grid Scheduling

Globus Architecture is being studied widely and nearly every scheduler is being designed to be compatible with it. So it is important to define certain parts of the Globus architecture, since these parts are used as a component of the scheduler architecture in most of the cases. In this section, the architecture of the Globus Resource Manager, which includes the allocation component, is described.

Globus uses Grid Resource Allocation Manager (GRAM) as the Local Resource Manager (LRM) [Foster, 1999]. Information Service and the database store the information about the resources including the nodes and the network. Globus Meta Directory Service (MDS) [Foster, 1997a] and Network Weather Service [Wolski, 2003a] are examples of services that can be used by an Information Service.

Figure 1-4 shows the Globus Architecture for Reservation and Allocation (GARA). GARA provides for dynamic discovery, advance and immediate reservation, and management of resources for networks, processors and disk memories. An application has to use the resources at multiple sites. At each site a Grid Resource Allocation Manager (GRAM) is used as the Local Resource Manager (LRM) [Foster, 1999]. A Co-allocation agent is used to map multiple resources to an application. Such an agent uses one or more GRAM(s). The resource Information Service can be provided by an MDS or a NWS system.

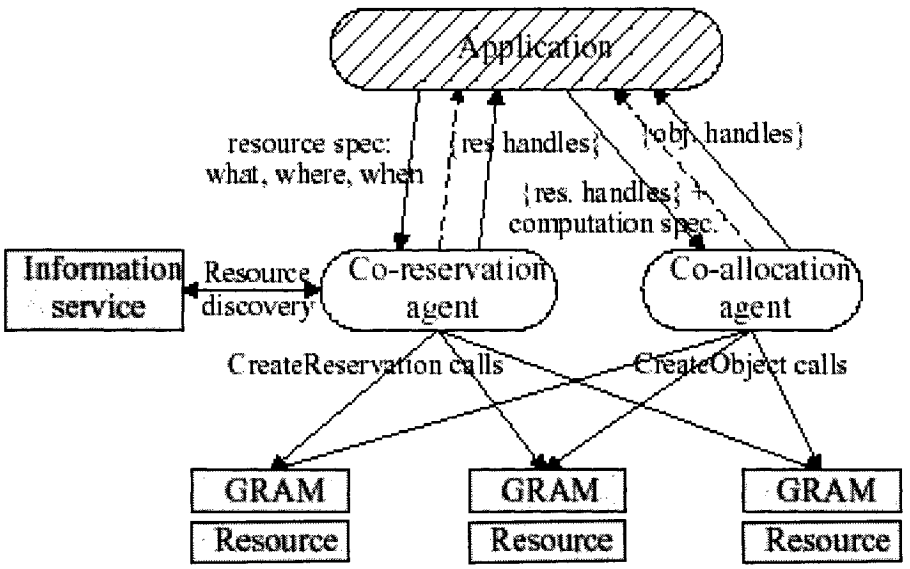

Figure 1-4 GARA Resource Management Architecture (Czajkowski, 1997, pp: 12)

Figure 1.5 provides the major components of GRAM. The Figure shows that a GRAM client obtains information from MDS. The GRAM Reporter sends information about the resource to MDS. The GRAM Client uses the Gatekeeper to load a task to the local resource set. The Gatekeeper uses the Grid Security Infrastructure (GSI) for authentication and other security services. The JobManager parses the tasks by using the Resource Scripting language (RSL) library and conveys it to the Local Resource Manager for final allocation to the local resources. The GRAM Client is able to obtain the status of the job through the Gatekeeper and the JobManager.

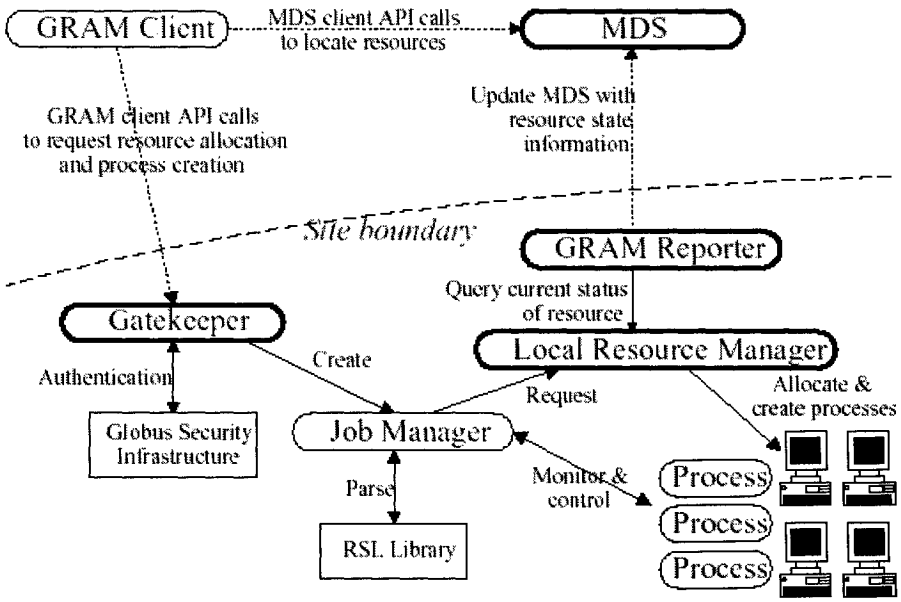

Figure 1-5 Major Components of GRAM Implementation (Foster, 1999, pp: 4)

In the new upgrades to the Globus tool-kit, a Lightweight Directory Access Protocol (LDAP) has been developed as the standard interface for both MDS and NWS. This makes it possible for GARA to use either MDS or NWS as the source for resource Information Service.

However the Globus system does not include a scheduler. The scheduler developed in this thesis may be used along with the Globus RMS or with the components being designed by our research group.

1.5 Globus Tool-kit: An Emerging Standard

The Globus tool-kit is becoming pervasive and is being developed by Global Grid Forum as a standard for the global grid.

Globus [Foster, 1998] tool-kit specifications provide for:

- Authentication.
- Discovery of currently available resources.
- Selection of a set of resources.
- Staging of the job to use those resources.

Thus the Globus tool-kit can be used to run a single job on a system, belonging to the grid.

1.6 Schedulers for the Global Grid

For a global grid, a scheduler will have the characteristics of being hierarchical and non-preemptive type.

An open worldwide market of grid resources has lead to a user having a choice of being able to use resources from a number of resource providers. Hence the scheduling process will have to consider the perspective of the user. However a resource provider can provide services at low cost to the consumers, only if the scheduler is able to maintain the throughput at a high level. Thus a scheduler is required to satisfy the goals of both the user and the service provider.

1.7 Problem Statement

In this thesis we will address the problem of designing a Genetic Algorithm based grid scheduler. We propose an algorithm, which satisfies the mutually conflicting goals of the users and the resource provider. The algorithm can handle multiple independent jobs with arbitrary precedence constraints among job's tasks and arbitrary processing times. The design is hierarchical and can be used at the lowest and the higher tiers.

1.8 Contributions

The contribution of this thesis is summarized as follows:

Development of a Genetic Algorithm based grid scheduler:

- It makes the best effort to satisfy the requirements of both the Resource Provider and the User conflicting goals.
- It has the ability to handle multiple independent jobs having arbitrary precedence constraints and arbitrary execution times.
- It is able to handle a wide variety of jobs efficiently.
- It supports dynamic availability of resources.
- The scheduler can be used at both the lowest tier or at the higher tiers of the hierarchical structure.

1.9 Organization of the Document

The literature survey is given in chapter 2. We start with the schedulers for clusters. We define the Globus grid architecture and the grid schedulers that have been developed. After describing the TITAN scheduler, the existing schedulers are compared. In chapter 3, we describe the design of our algorithm. Since it is based on the theory of Genetic Algorithm (GA), we describe the general process of search for a solution using GA, before proposing our design. Chapter 4 includes testing, validation and an empirical selection of parameters, based on experimentation. By using these parameters, we test the algorithm with multiple standard jobs and we present the results of these tests.

Chapter 2

2 Current Schedulers for Computational Grids

2.1 Introduction

The idea of connecting the computers in a network so that they can communicate with each other was practically implemented in 1969 by Len Kleinwck in his laboratory at UCLA. The concept of time-sharing in a computer was developed in the 1970s. This allowed multiple jobs to be processed by multiple virtual computers in parallel.

Parallel-processing is the use of multiple CPUs to solve problems, which require a computing power, not easily available in a single CPU machine. The parallel processing can be through multi-CPUs on a single machine or through multiple autonomous computers connected in parallel. Since the seventies, compute-processors have been connected in parallel to provide adequate computing power for grand challenge problems, for meeting the computing needs of a large number of users and for high availability. Such a parallel system of computing machines can be called a cluster, if with appropriate middleware; the system can be used, as a whole, to process a single or multiple application(s) efficiently. The cluster may consist of

SMPs, massively parallel machines, mainframes or commodity machines. A homogeneous cluster consists of all machines with the same hardware and software configuration. On the other hand a heterogeneous cluster may consist of a variety of hardware and software platforms.

With the worldwide availability of Internet, an application can be processed through local computers working jointly with remote computers. So a heterogeneous distributed cluster is obtained through an interconnection of geographically distributed and heterogeneous computers, connected through a network [Liu, 2004]. Computers in a cluster are dedicated to the jobs, required to be processed by the cluster and they are tightly coupled. A grid consists of a set of non-dedicated and geographically distributed machines, connected through high-speed network. A grid may be required to handle a wide variety of applications.

For scheduling of tasks on a cluster, a number of heuristic algorithms have been studied during the last three decades.

Grid scheduling is the process of developing a schedule for mapping tasks of jobs, such that the objectives of the scheduling process are satisfied. The first set of grid scheduling algorithms was developed as generalizations of cluster scheduling algorithms.

In this Chapter we shall begin Section 2.2 by giving briefly the evolution of the idea of a grid, Section 2.3 will define the characteristics of a cluster. Section 2.4 describes three approaches, which have been used in the cluster scheduling heuristic algorithms. Two of the three have been found to be of direct use in developing grid schedulers. In Section 2.5 we shall describe a number of research projects relating to grid scheduling. A comparison of the well-known grid schedulers is given in a tabular form in Section 2.12.

2.2 Evolution of Grid Computing and Grid Schedulers

In 1997 Foster discussed the idea of multiple nodes at geographically distributed locations [Foster, 1997a]. At the end of 1997, Foster and Kesselman [Foster, 1997b] presented Globus, a toolkit. In the paper, the authors said that the toolkit was for networked virtual supercomputers. The Globus toolkit was a part of their work on the development of Adaptive Wide Area Resource Environment (AWARE). In 1998, Foster and Kesselman [Foster, 1998] borrowed the term grid from electrical power systems and applied it to computational grids.

In 2000, Buyya [Buyya, 2000a] developed Nimrod-G as a scheduler based on the economic principles. He suggested that grid applications could be scheduled on the basis of the cost of available resources and the needs of the application.

In 2000 IEEE Computer Society's Technical Committee on Cluster Computing held its first Grid 2000 workshop in Bangalore. In 2001, researchers from all over the world met in Amsterdam to form the Global Grid Forum (GGF). [Web-GlobalGrid].

In 2002 Condor-G [Frey, 2002] for grids was developed on the basis of Condor, which had been developed for efficient management of compute-resources under a single domain in 1988. Condor-G attempted to maximize the use of grid resources. Scheduling, through Condor-G, satisfied the goals of the resource providers. However it did not take into account the ease of use for a user. It also failed to work for QoS, as perceived by the users for their applications. Berman et al presented in 2001 the GrADS [Berman, 2001] scheduler. Its objective was to provide ease-of use to a user of grid services. AppLeS [Berman, 2003] is a scheduler, which explicitly satisfies the goals of an application.

Today every country is trying to use the computational and data grids to strengthen research in high-performance systems. Many large grid projects, which connect universities and research laboratories, are being built in all the developed countries.

2.3 Cluster Scheduling

"Cluster is a widely-used term meaning independent computers combined into a unified system through software and networking. At the most fundamental level, when two or more computers are used together to solve a problem, it is considered a cluster. Clusters are typically used for high availability for greater reliability or High Performance Computing (HPC) to provide greater computational power than a single computer can provide." [http://www.beowulf.org/overview/].

A cluster consists of machines connected together through a high-speed network. A cluster consists of dedicated and tightly-coupled workstations. A cluster is managed by a single administrative entity. A cluster has a server, which interacts with the users called clients. The server also has information about the computer nodes called the agents in the cluster. A client may be connected to the server through the Internet. The client has to inform the server about the requirements of the application. The inter-dependency of the processes of the application has also to be specified, by the client, either through a script or through a GUI interface for application building. A cluster can run both sequential and parallel jobs.

According to [Abawajy, 2003], "The key to making cluster computing work well is the middleware technologies that can manage the policies, protocols, networks, and job scheduling across the interconnected set of computing resources." The two main goals of cluster scheduling are to maximize the utilization and throughput.

These schedulers allow the administrator to set the following parameters:

- To set multiple queues for different job classes.
- To set priorities and scheduling policies for each queue.
- To treat interactive jobs differently from non-interactive jobs.

2.4 Grid Scheduling Approaches

First-Come-First-Serve (FCFS)

The job arrival times and the arrival rates may vary for a grid. A grid scheduler that follows the FCFS policy will allocate the jobs to compute nodes in the order of their arrival. The jobs, which arrive when all the resources are in use, will wait in a Wait queue. "This strategy is known to be inefficient for many workloads as a large number of jobs waiting for execution can result in unnecessary idle time of some resources" [Schwiegelshohn, 2000b]. Zhang, [Zhang, 2003] have used three approaches to improve the performance of schedulers.

Backfilling

A scheduling process would map tasks to compute-nodes and put the tasks in a queue according to a priority set by it. Backfilling is the method, which attempts to use a node, during its unutilized time gap, for processing lower priority (smaller) tasks in the queue, rather than keeping the nodes idle. The node may have been allocated, after the gap-time, to a higher priority task. If the node is not released by the lower priority tasks, the mapping according to the schedule cannot be possible. To avoid such an eventuality, backfilling requires that the lower priority task should be completed before the higher priority tasks that have been mapped by the scheduler to the same node, are scheduled to start.

If a task requires more time than the estimate provided by the user, the job is terminated. Such a policy will ensure that the users will not under-estimate the time required for execution of the tasks, while submitting the metadata about the job.

Gang scheduling or Coscheduling

For implementing gang -scheduling, the available machines are time sliced to provide at least as many virtual machines as the number of tasks that can be run in parallel. In some cases the same task may be run in multiple virtual machines. The number of virtual time slices created on a machine is called the Multi-Programming Level (MPL) of the system. This technique reduces wait time. But it increases apparently the execution time. The advantage of the gang scheduling

method is that its performance does not depend upon the accuracy of the estimates of task execution times. The disadvantage is that it introduces context-switching overhead.

Migration

In this approach, if a compute node is overloaded, some of the tasks in its waiting queue can be migrated to those processors, which are lightly loaded. Migration provides flexibility of adjusting the schedule to avoid fragmentation. Co-allocation in space is important in some architecture to guarantee proper communication among tasks.

[Zhang, 2003] have shown that the three approaches jointly improve the scheduling process in clusters. A grid uses the idea of space sharing and nearly all grid-scheduling algorithms use the concept of backfilling.

2.5 Existing Grid Schedulers

The grid scheduling systems are at an early stage of development. As the use of grids progresses from research grids to a worldwide grid, resource-scheduling systems will have to become more efficient.

The grid scheduling systems of today are based on the scheduling systems for clusters developed during the eighties and the nineties. PBS in its various versions has been developed over the last decade. It aims at minimizing the overall turnaround time by using techniques like backfilling.

Condor was developed in 1998 [Litzkow, 1988] to manage local resources. Even though the managed resources are non-dedicated and heterogeneous, a compute-node is used for remote scheduling only when a user is not using it directly. As soon as a user begins using a compute-node, the remotely scheduled task is migrated to another node. Condor aims at high throughput computing. It makes no effort at load balancing and it assumes that the jobs are always available in the queue.

When Condor was being developed, Foster was developing [Foster, 1997a] the concept of multi-cluster systems, which evolved into the idea of a Grid in 1998 [Foster, 1998].

In 1999 Maheswaran et al [Maheswaran, 1999] developed the scheduling heuristics for independent jobs for use on a cluster. Abramson [Abramson, 2000] proposed NIMROD scheduler. It used the economic model for parameter-sweep applications. However NIMROD assumes that all the resources are under the control of NIMROD.

By 2001 Globus tool-set had become known and the Globus team started working with the Condor team to develop Condor-G [Frey, 2002]. However the end-user is required to specify a list of GRAM servers, through which the users want their job to run. Condor-G does not try to use the best resources available for executing the job.

Buyya generalized NIMROD to work with Globus tool-kit. He christened it as NIMROD/G [Buyya, 2002c]. NIMROD/G requires users to specify a deadline for each job. The jobs are prioritized, based on the deadline. It uses the Globus Security Infrastructure for authentication. But both NIMROD and NIMROD/G also continue to require complete control of resources.

Two other major projects on scheduling architectures were developed during 2000-2002 at San Diego Super Computing center and at University of California. AppLeS [Casanova, 2000] at SDSC was designed for parameter sweep applications such that many simulations may use the same input. Moreover an output file from one simulation can be used as the input file for the next. Thus AppLeS optimizes input-output operations.

GrADS [Berman, 2001] attempts to schedule parameter sweep applications without knowing about the estimated runtime. It stops long-running jobs and starts shorter ones to improve the turn-around time.

In 2000 Buyya, in a theoretical work [Buyya, 2000b], enumerated heuristic methods, which may be used to get the best possible schedules out of the large solution space for even middle sized systems. However it was in 2003 that the first major project at Warwick University reported some results [Spooner, 2003a]. The project is called TITAN. The scheduler in TITAN architecture attempts to obtain optimum scheduling through Genetic Algorithm.

Many of the available grid schedulers are centralized. They assume that the jobs are received at a central point and the scheduling process is controlled by the scheduler, which has complete information about all the machines. Condor, Nimrod and GrADS are examples of the centralized schedulers. However it is distributed architectures for resource management, which can be scalable to global levels.

The schedulers have been developed from the perspective of efficient resource usage [Condor and Condor-G [Frey, 2002]) or from the perspective of applications (AppLeS [Berman, 2003], GrADS [Berman, 2001]) or from the perspective of a grid as market (Nimrod and Nimrod-G [Buyya, 2000a]).

Since the goals of a user and the service provider are not fully compatible, the available schedulers are either user-centric or resource provider-centric. The economy-based schedulers mimic the market model, with the assumption that the market should benefit both the user and the service-provider. For meeting the diverse goals, required by a user and a service-provider, the TITAN architecture, has developed, at the leaf level, a genetic scheduler.

2.6 Condor

The Condor High Throughput Computing System is a combination of dedicated and opportunistic scheduling. "Opportunistic scheduling involves placing jobs on non-dedicated resources under the assumption that the resources might not be available for the entire duration of the jobs" [Wright, 2001]. Condor respects site autonomy. It does not provide co-allocation.

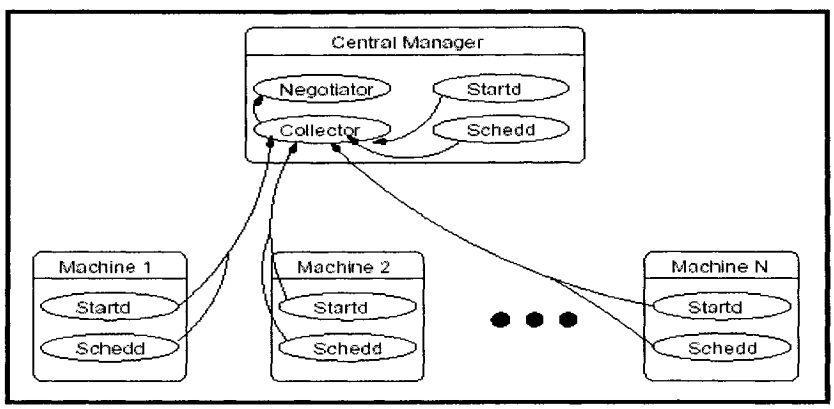

Figure 2-1 Architecture of a condor pool with no jobs running (Wright, 2001, pp: 4)

As shown in Figure 2-1, the architecture of Condor uses a daemon called the collector, for centrally maintaining the list of Condor resources. Periodic updates are sent to the collector by Condor daemons on computing machines. The collector runs on the machine called the Central

Manager. The Central Manager contains a negotiator, which tries to find a match between resource requests, and resource offers [Wright, 2001]. Each computational resource within the pool is represented by a demon called startd. Another demon called Schedd, handles the user jobs. All the jobs are submitted to Schedd. The main functionality of Schedd is to maintain a job queue, to publish resource request and to negotiate for available resources.

Condor, assumes that jobs, to be processed by a grid, would be independent and non-real time jobs. Resources are assumed to be independent workstations. The information about the computing resources only specifies the availability of the resource. When it is used for grid processing, the resource is not to be used for processing local jobs. Thus it is not time-shared. However as soon as a local job is loaded on a resource, the job, brought through the grid, will be pre-empted (i.e. it would be moved to another resource available in the pool).

Checkpointing is used for storing the state of processing of every job so that, if required, the job can be moved. Condor does not take into account the overhead of transferring a job. Condor scheduling is designed to increase the utilization of workstations within a single domain [Wright, 2001].

Both Condor and Condor-G use matchmaking algorithm, which allocates an available resource to a job. If the resource should become unavailable during processing, the job is migrated to another compute-node that may be available.

2.7 Condor-G

Condor-G is a combination of Condor and Globus toolkit [Frey, 2002]. The Globus toolkit supports resource discovery and resource access in multi-domain systems. Condor-G allows a user to harness multi-domain resources as if they all belong to a single domain. It also uses authentication, authorization and secure file transfer facilities provided by the Globus tool-kit.

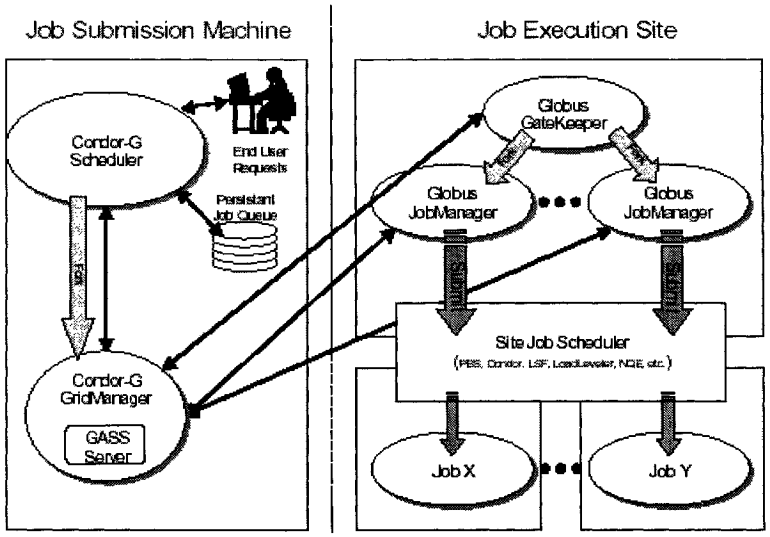

Figure 2-2 Remote execution by Condor-G on Globus-managed resources (Frey, 2002, pp: 4)

As shown in the Figure 2-2, Condor-G uses Globus GASS (Global Access to Secondary Storage) file server, G.S.I (Grid Security Infrastructure) and GRAM (Grid Resource Allocation and Management) protocol. The user accesses Condor-G scheduler from the user desktop. A local Grid-Manager daemon is created along with a GASS server. The Grid Manger gets authenticated with the Globus tool-kit at the job execution site. A Globus Job Manager daemon is created at the job-site. The Grid Manager and the Job Manager cooperatively get the job processed by using the available grid resources.

The Condor-G scheduler at the user's desktop maintains a persistent job queue, so that failure of any part of the Grid would make available to the user the status of the job and state of the processed part up to the last checkpoint.

2.8 AppLeS Scheduler

AppLeS stands for Application Level Scheduler [Berman, 2003]. It has been designed for meeting performance goals, which are specified by the application. Each application has its own scheduling agent. The agent monitors available resources and generates a schedule for the application.

The resource discovery and prediction is implemented through Network Weather Service [Wolski, 1999b]. The user has to specify the necessary information about the application and the performance goal that the scheduler attempts to meet.

A number of possible schedules are developed along with the expected performance index for each case, as a part of the process of obtaining the final schedule. The schedule, which maximizes the performance goal, is selected and implemented. The scheduler adaptively learns from every cycle of implementation to refine its working.

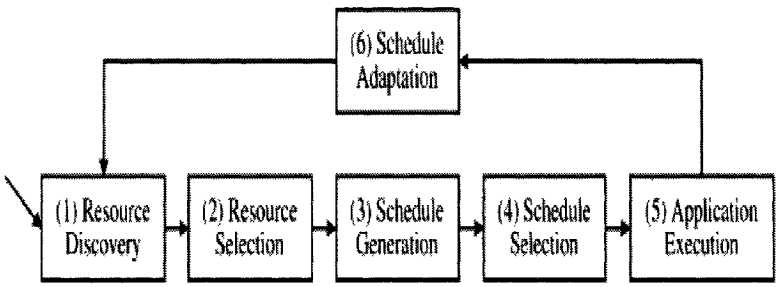

Figure 2-3 Steps in the AppLeS methodology (Berman, 2003, page: 2)

AppLeS Scheduler uses the architecture of a central server and multiple agents. Jobs, which can be partitioned into as many parts as the number of available compute resources, are considered. The scheduler works for a class of problems, which are flexibly partitionable. For long running jobs, which require file sharing, a heuristic XSufferage has been developed.

"The Sufferage heuristic is based on the idea that better mappings can be generated by assigning a machine to a task that would "suffer" most in terms of expected completion time if that particular machine is not assigned to it." [Maheshwaran, 1999]. If the calculation of sufferage value takes into account the data transfer costs, the heuristic is called Xsufferage.

2.9 Nimrod-G Resource Broker

Nimrod-G was developed at Monash University, Australia. Nimrod mimics the model of a market for a single domain [Buyya, 2000a]. Nimrod-G has been designed by modifying Nimrod for operation with Globus tool kit as shown in figure 2-4.

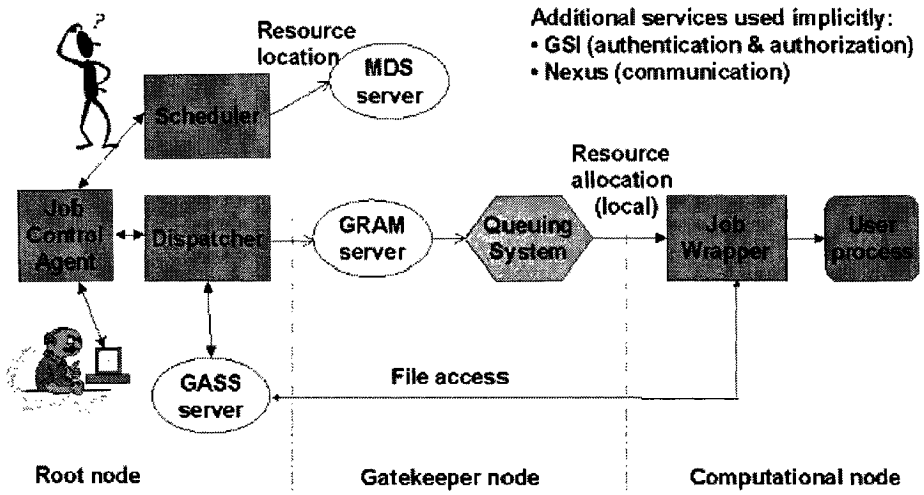

Figure 2-4 Nimrod/G and Globus Components Interactions (Buyya a, 2000, pp: 4)

The resource discovery and job allocation over a grid have been implemented through the Globus tool-kit whereas Nimrod handles the market mechanism.

Each Nimrod application has a specified budget and a deadline, before which it should be completed. Each resource has a price, which must be paid if the resource is to be used. The scheduler is designed to complete the application within the deadline at the minimum cost. Similarly each resource tries to maximize the gain by providing its services. If the budget or the deadline should be exceeded, the user is informed about it. Every application in Nimrod-G has an associated Job Wrapper, which acts as a mediator between the resource and the application. Mediator is also used for sending the required information to the Resource Accounting system.

Nimrod-G is part of a framework called Grid Architecture for Computational Economy (GRACE). GRACE includes a global scheduler called a broker. The broker works with other components like bid-manager, directory-server, and Globus tool-kit to maximize the performance goals.

2.10 GrADS Scheduler

The Grid Application Development Software (GrADS) project aims at making it simple for users to use grid resources [Berman, 2001]. After the application has been prepared in the GrADS program preparation system, it is delivered to the Scheduler/Service Negotiator system. The scheduler is a part of the GrADS execution environment. To improve the performance of a new application, GrADS can preempt an executing application, if the resource is overloaded.

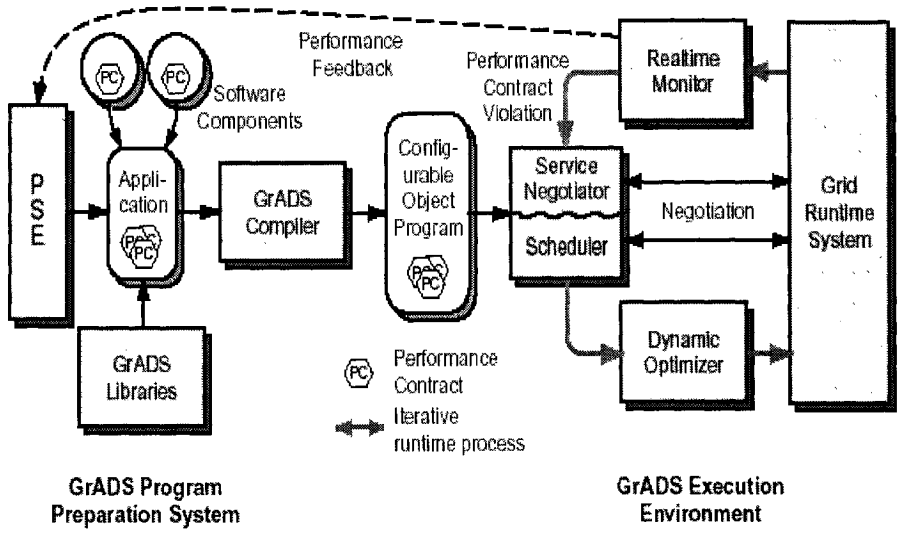

Figure 2-5 GrADS Program Preparation and Execution Architecture (Berman, 2001, pp: 8)

When the object containing the job with a performance contract is delivered to the Scheduler/Service Negotiator, it will broker the allocation and scheduling of grid resources for the job. Thereafter the dynamic optimizer is activated to adapt the program to the available resources. It will also insert sensors for monitoring the status of the job.

The real time monitor verifies that the requirements of the performance contract are satisfied. In case there is a violation, the execution may be interrupted. Either the optimizer may adapt the program or new resources may be negotiated or both the steps be taken to ensure compliance with the performance contract. Thus the closed loop system of GrADS scheduler ensures Quality of Service (QoS). To estimate the available resources when the job is to be executed, GrADS uses a predictive algorithm.

Grid Application Development Software (GrADS) Scheduler [Casanova, 2003] deals with independent jobs only. The Candidate Machine Group algorithm prepares ordered lists of machines and matches metadata of job requirements with the ordered list to obtain a mapping of resources to jobs.

2.11 Genetic Algorithm for Grid Schedulers

In the early 1970s, John Holland introduced the concept of genetic algorithms. Genetic algorithms use biologically derived techniques such as inheritance, mutation, natural selection, and recombination. The goal is to search for nearly the best solution in an efficient manner, out of a solution space, which is very large. [Holland, 1975]

Genetic algorithms are used to find approximate solutions to difficult-to-solve problems through application of the principles of evolutionary biology to computer science.

The grid scheduling algorithms can be designed to satisfy the goals of the resource provider or the users of grid services. AppLeS or GrADS are examples of such schedulers. Genetic algorithms are used to determine efficiently schedules, which attempt to meet the goals of both the resource provider and the user. In addition the weights can be used to change the behavior of the scheduler on-the-fly. A GA-based scheduler is also able to adapt to minor changes in the problem space, without using up a great deal of overhead.

2.11.1 TITAN GA-based Scheduler

The TITAN architecture "employs a Performance Analysis and Characterization Environment (PACE) for the predictor and task distribution brokers to meet user-defined deadlines and improve resource usage efficiency" [Spooner, 2003a].

Iterative heuristic algorithms are applied for performance prediction for job schedulers in Grid environments. Workload managers, Distribution Brokers and Globus Interoperability providers comprise the TITAN system's hierarchy, with Globus forming the highest level in the hierarchy. The lowest level consists of schedulers for managing physical resources as shown in Figure 2-6.

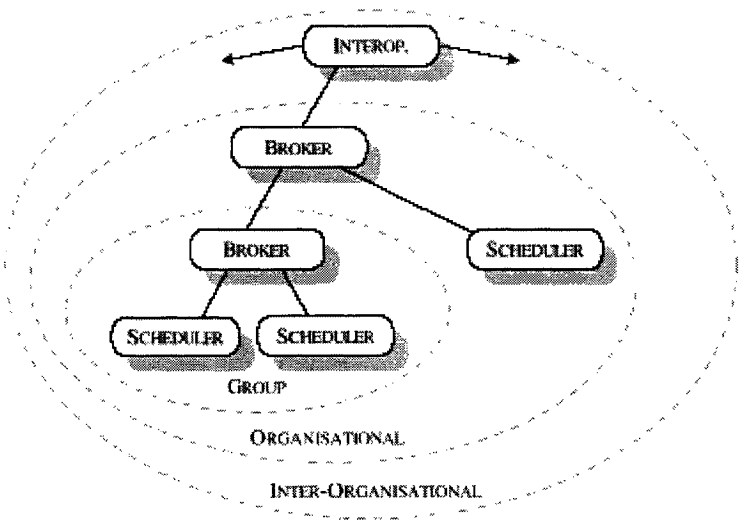

Figure 2-6 The TITAN architecture (Spooner, 2003a, pp: 3)

The Genetic Algorithm approach has been found to be less sensitive to the number of processing nodes and to minor changes in the characteristic of the resource. Moreover it can take into account a large number of objectives of the resource system. The genetic algorithm is able to converge fast to balance the three objectives of makespan, idle time and the QoS metric of deadline time.

System architecture of the Scheduler

The GA-based local scheduler in the TITAN system is for a homogeneous cluster. The scheduler uses the Globus (MDS) to learn about the available resources. Advertising & Discovery Agents (ADA) and PACE are able to predict the metadata about the job so that the scheduler can efficiently map the jobs to available resources. However PACE is able to handle relatively simple type of jobs.

Types of Jobs

All the tasks must run in parallel on a pre-defined number of machines. Moreover all the tasks should be of the same duration and should run simultaneously. No dependencies among the tasks can be handled by the system.

Genetic Algorithm Representation

The scheduler uses Chromosome, consisting of a permutation of the given set of tasks.

$$\ell_A = [T_1, T_3, T_5, T_2, T_0, T_6, T_4, T_7]$$

Figure 2-7 Chromosome Representation (Spooner, 2002b, pp:8)

Crossover Operation

The crossover process consists of merging a portion of one chromosome with the remaining portion of a second chromosome, and then re-ordering to eliminate illegal duplicate values of the same tasks as shown in the figure 2-8.

$$\ell_A = [T_0, T_3, T_5, T_2, T_6, T_4, T_1]$$
$$\ell_B = [T_5, T_4, T_3, T_2, T_1, T_3, T_6]$$

$$\ell' = [\ell_{A.0}, \ell_{A.1}, \ell_{A.2}, \ell_{B.3}, \ell_{B.4}, \ell_{B.5}, \ell_{B.6}]$$
$$= [T_1, (T_3), T_5, T_2, T_6, T_7, T_0, (T_3)]$$
$$= [T_1, T_4, T_5, T_2, T_6, T_7, T_0, T_3]$$

Figure 2-8 Crossover Operation (Spooner, 2002b, pp: 8)

Mutation Operation

The mutation process involves swapping randomly two of the tasks in a chromosome as shown in figure 2-9.

$$\begin{aligned} I^{M} &= [T_0, T_4, T_5, T_2, (T_1), T_3, (T_6)] \\ &= [T_0, T_4, T_5, T_2, T_6, T_3, T_1] \end{aligned}$$

Figure 2-9 Mutation Operation (Spooner, 2002b, pp: 8)

Fitness Function

The fitness function takes into account makespan, idle times of nodes and the deadlines of a task.

Experimental Results

The experiments, reported for the scheduler, have been conducted using 32 jobs and a 16-node cluster. Population sizes of 20, 40 and 60 were used. It was found that population size of 40 converges after about 1000 generations. Upto generation 600, makespan dominates. Beyond 600, deadline drives the fitness function. A Pentium 800 MHz PC was able to compute 100 iterations/sec.

2.12 Comparison of Schedulers

The characteristics of the well-known schedulers, available in the public domain, are given in Table 1. Schedulers can be compared on the basis of the following features.

- User-Centric
- Service-Centric
- Metrics
- Pre-emptive
- Multiple-Domain
- Application Type
- Algorithm

	GrADS	Condor-G	Nimrod-G	AppLeS	TITAN
User-Centric	YES	NO	YES	YES	YES
Resource-Provider Centric	NO	YES	NO	NO	YES
Metrics	Minimize Total Turnaround Time	Maximizing Throughput	Minimize the Cost; Meet the Deadlines	Minimize Execution Time	Minimize the Makespan and IdleTime; to meet the Deadlines
Pre-emptive	YES	YES	NO	YES	NO
Multiple-Domain	NO	YES	YES	NO	YES
Application Type	Iterative /mesh-based	Parallel	Task Farming Application (Parameter studies)	Task Farming Application (Parameter studies)	Parallel – rigid jobs
Algorithm	Candidate Machine Group	Matchmaking	A Deadline and Budget Constrained Cost-Time Optimization Algorithm	XSufferage	▪ Genetic-Algorithm ▪ Deadline-Sort ▪ Deadline Sort with Node Limitation

Table 1 Comparison of Schedulers Characteristics

2.13 Conclusion

The chapter describes the well-known schedulers and compares their characteristics. The architecture has been described only to the extent it is relevant for the scheduling algorithms. In the next chapter, a description of the algorithm developed, as a part of this thesis, has been presented.

Chapter 3

3 A new GA Based Scheduler

3.1 Introduction

Every grid scheduler is designed for specific resource management architecture. The architecture describes the generalized method of acquisition of meta-data of the jobs and collection of information about the available resources. Scheduling is the process of mapping the tasks of a set of jobs to compute-nodes efficiently. The architecture includes the methods of staging the jobs according to the schedule, of monitoring the processing of tasks and of rescheduling the tasks, if required.

This thesis deals with highly scalable distributed resource management architecture for the global grid. The main component in the proposed architecture is the Genetic Algorithm based scheduler. We shall be able to show that the scheduler is able to use the available resources efficiently, while satisfying competing and mutually conflicting goals. The grid workload may consist of multiple jobs, with quality-of-service constraints. A Directed Acyclic Graph (DAG) represents each job. Each task may have arbitrary precedence constraints and arbitrary processing time. The scheduler has been designed to be compatible with other tools being developed by our grid research group.

This chapter presents the design and implementation of the scheduler. The scheduler attempts to minimize make-span, idle time of the available computational resources, turn-around time and the specified deadlines by the users. The architecture is hierarchical and the scheduler is usable at either the lowest or the higher tiers. It can also be used in both the intra-grid of a large organization and in a research grid consisting of large clusters, connected through a high bandwidth dedicated network.

Section 3.2 describes the architecture of the resource allocation system in which the scheduler is expected to operate. A grid scheduler should be able to handle a wide variety of jobs. Most of the published work on schedulers shows tests with a specific type of problem, as discussed in section 2.5. Standard jobs with up to 3000 tasks with arbitrary precedence constraints have been

used. These are briefly described in section 3.4. The objectives of the grid scheduling process determine the fitness function, which is described in section 3.8. Section 3.11 describes the process of generation of new population, including the process of Roulette wheel method, developed for the scheduler. The processes of crossover, mutation and elitism are described in the next three sections. The chapter ends with a description of the complete simulation process and a statement of the different modules, which constitute the scheduler.

3.2 The Hierarchical Architecture of Resource Allocation Systems

A global grid would have a dynamic, geographically distributed, heterogeneous and non-dedicated set of resources. The resources can be distributed over a set of domains of variable size. Each domain can be called as a Resource Allocation System (RAS). The RASs can be hierarchically organized.

Each RAS, at the leaf level, will be directly dealing with available resources. So it has information about the resources. During negotiations with the users, the scheduler of the RAS can generate the schedules for all the jobs, for which meta data is available. The scheduler is able to work out the extent to which it is possible to meet the goals specified by the user and the resource provider. If the requirements of some users are met, they can decide to have the jobs processed through the RAS. On receipt of the actual jobs and their meta-data, it may have to recalculate the schedules, if the deals for some of the jobs do not work out. Then the Allocator allocates the jobs to the resource set.

In case RASs are organized hierarchically, the RAS at the highest tier can negotiate with the users, on the basis of the resource set information by the RASs at the lower tiers.

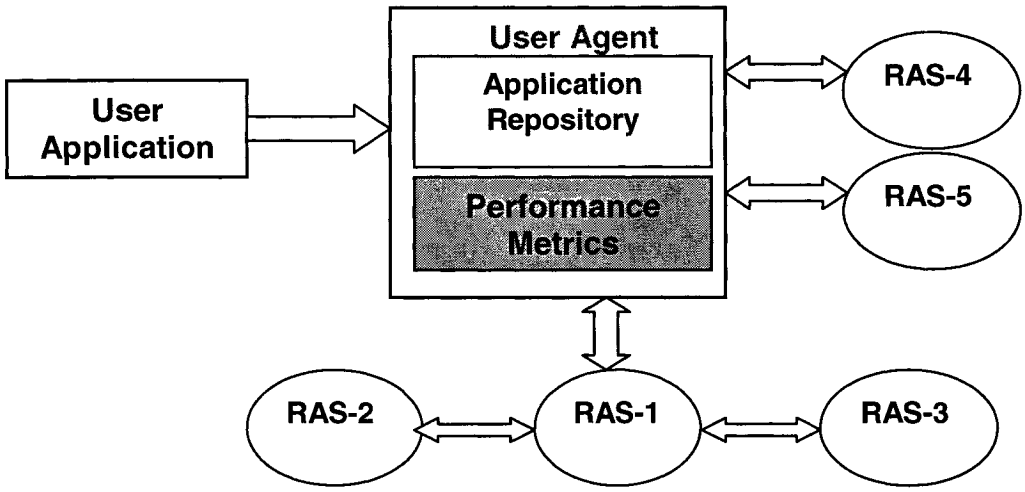

Figure 3-1 A Hierarchical Architecture of Resource Allocation System (s)

The scheduler at the highest tier works out the schedules and sends all the jobs or some jobs to a RAS at the leaf level. The RAS at the leaf level is also provided with the meta- data about each full job. If the need for rescheduling the schedule arises, due to the non-availability of some compute-node in the domain of the leaf level RAS, the local scheduler can generate. If the rescheduling schedule satisfies the user requirements, it is required to update the resource availability data at all the connected RASs.

3.3 A Resource Allocation System

A resource Alloçation system consists of components, which are able to negotiate with the users and obtain the jobs. The jobs can then be staged to the resources for processing. The main component in an RAS is the scheduler.

The job acquisition system consists of the Negotiating Agent and the associated job queue and database for the meta-data of the jobs. The Negotiating Agent obtains the meta-data of the job and the requirements of the user. The Resource Information repository contains information about the free time available for each compute-node in the resource set. It also has information about the characteristic of each compute node. The scheduler uses the meta-data of the jobs along with the information about the availability of resources to generate the schedules for each

task. It develops mappings of the tasks of jobs to the available resources in such a way that the requirements of the user and the goals of the resource providers are satisfied. It also obtains the values of the turn-around time for each job and information about whether it would be able to meet the specified deadline. For the resources, it works out the session completion time, called the makespan, and the idle time of the nodes. The negotiating agent conveys the information, generated by the scheduler about a job, to its user. If the user finds that the RAS is able to meet the user requirements, the job can be given to the RAS. If the acquired jobs are less than the jobs that were considered for scheduling, a new schedule can be generated.

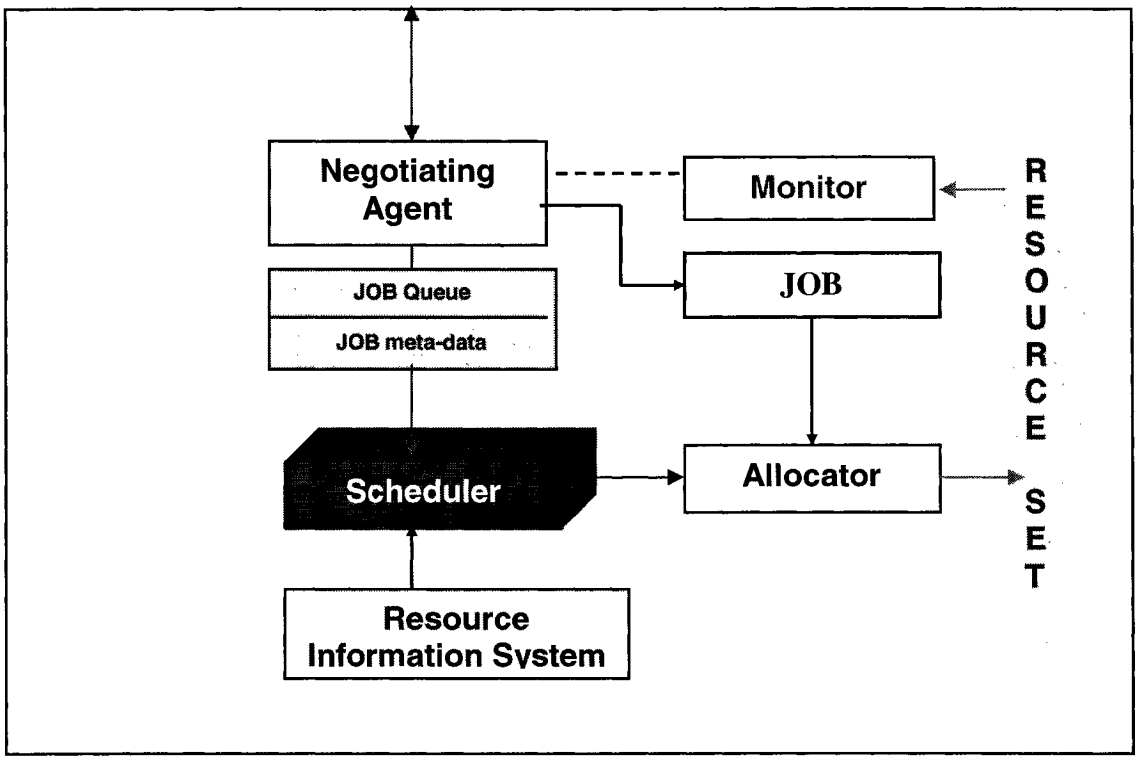

Figure 3-2 A Resource Allocation System

The Allocator in the RAS uses the mapping, generated by the scheduler, to stage the tasks of all the acquired jobs to the available resources. The Monitor obtains the final results, when a job has been processed and sends them to the user through the negotiating agent. If a resource, on which a task has been allocated, should become unavailable during processing, a scheduler may reschedule the new schedules. If these satisfy the requirements set by the users, the allocator again allocates the remaining tasks to the resources. If it is found that the revised schedule is not able to meet the user requirements, the negotiating agent informs the user.

3.4 The Directed Acyclic Graph (DAG) Model for a Job

A grid is supposed to handle a wide variety of jobs. Many users will have their own objectives. Common objectives can be minimization of the turn-around time for their jobs. Some jobs can have stringent deadlines. Similarly a resource provider would want a high throughput and a minimum idle time for compute-nodes. A scheduler has to map the tasks to resources in such a way that the objectives of both the user and the service provider can be satisfied. As discussed in the previous chapter, most of the schedulers have been shown to handle a class of problems only. The parameter sweep problem has been solved by a number of available schedulers.

[Kasahara, 2002] has compiled a set of standard jobs for testing and comparing scheduling and load balancing algorithms on clusters. For this purpose, the set consists of jobs described by Standarad Task Graphs (STG). Each node of a STG graph represents a task of the job. The weight of the node specifies the computational time required by the task. An edge of the graph shows the dependency of one task on the other. Figure 3-3 shows an example of the DAG model for a job.

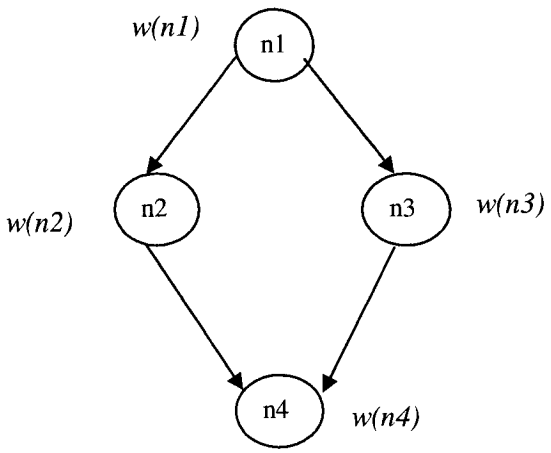

Figure 3-3 A DAG Model for a Job

STG job set has been compiled for comparing the heuristic algorithms for scheduling "n" computational tasks of a job on "m" processors. The tasks have arbitrary precedence constraints and arbitrary processing time. The motivation for generating STG set, according to [Kasahara, 2002] was as follows:

"The performance of scheduling algorithms has been either evaluated using randomly generated graphs or by task graphs modeled from actual application programs. However these task graphs are not typically available to other researchers. Therefore fair performance comparisons of the algorithms under the same conditions has been impossible. To allow researchers to evaluate their algorithms under the same conditions, we propose a Standard Task Graph set, covering previous task graph generation methods."

A parameter sweep type of job can be considered a degenerate case of a STG job. If therefore a scheduler can be designed, developed and tested using a set of STG jobs, it should be able to handle all types of jobs.

Each STG job can be represented by either a graph of the type shown in Figure 3-3 or by a file of the type shown in Figure 3-4. Since readable graphs of such large jobs are difficult to draw on even reasonably large sheets of paper, Kasahara Laboratory provides only files of STG jobs of 50 to 3,000 tasks. Our scheduler accepts meta data about jobs through such files.

Job File

total no of tasks

no	processing time	no of parents	parent1	parent 2	.....	parent n
0	0	0				
1	10	1	0			
2	20	1	1			
3	40	2	1	2		
.	.	.	.	.		.
.						
.						
n	p(n)	parents(n)	.....			

Figure 3-4 The Job File for a Standard Task Graph (STG) - (DAG model)

3.5 A Genetic Algorithm based Scheduler

The solution space for mapping of jobs to a cluster of a smaller number of dedicated compute-nodes is so large that it has been possible to develop cluster schedulers only on the basis of some heuristic algorithm. The grid scheduler is required to map dynamically available and non-dedicated resources to the tasks of the jobs. The compute-nodes may be large in number and may be distributed geographically. Moreover a grid is required to handle a wide variety of jobs. Thus the solution space for a grid scheduler is likely to be much larger than that of any cluster.

A GA based search can obtain a near-optimum value in a reasonable amount of time, even when the solution space is very large.

In a grid environment, the goals of the user and the resource provider can be conflicting i.e. the goals are such that when one attempts to improve one performance criterion, the other performance criterion may be degraded. A GA based search can try to reconcile conflicting objectives by attempting to satisfy requirements of all the stakeholders.

Hence a GA based scheduler for grid applications has been chosen for the proposed architecture. Grid scheduling is essentially an optimization problem. It is required to schedule jobs, while satisfying multiple objectives. Weights can be used with every objective to change the behavior of the scheduler on-the-fly. Thus it can allow the scheduler to prioritize on multiple objectives. A GA-based scheduler can adapt easily to minor changes in the problem space.

Under TITAN project [Spooner a, 2003], a GA based grid scheduler has been designed. However it is able to handle only highly degenerate cases of the generalized STG jobs.

3.6 The Genetic Algorithm

The genetic algorithm begins with the first generation of a population of individuals, called chromosomes. For each chromosome an appropriate fitness function is defined. For each chromosome of the population of the new generation, the value of fitness is calculated. A convergence criterion, which determines whether the required solution has been obtained, has to be defined for each problem. The criterion is a fitness function. We verify whether the convergence criterion has been reached. If it is, the process ends and the best solution,

obtainable through the genetic algorithm, is available. Otherwise the algorithm generates a new population of chromosomes. This is called the next generation. For generating the new population, the processes of selection, crossover, mutation and elitism are used.

After obtaining the new population, the fitness value for each chromosome of the new population is calculated. The standard deviation is also calculated and it is used as the criterion for convergence. Figure 3-5 shows the algorithm.

```
Genetic Algorithm ()
{
        Initialize population;  /*an initial population of permutation randomly generated*/
        Evaluate population; /*check fitness of each chromosome against fitness function*/
        do until termination criterion not reached
        {
                        Generate the next population;
                        /* apply genetic operators */
                                Perform crossover operation;
                                Perform mutation   operation;
                        Evaluate population;
        }
    end do
}
```

Figure 3-5 Genetic Algorithm

3.7 The Chromosome

A Chromosome represents a schedule of independent jobs, submitted to the RAS. In general each job is represented by a DAG. Thus each job has many tasks, with multi-level precedence constraints. A new chromosome can be generated by permuting the jobs in a given schedule. The initial population is generated by a random permutation of jobs as shown in figure 3-7.

Position	1	2	3	4	5	6
	J3	J6	J5	J2	J1	J4

Figure 3-6 Representation of a Chromosome

J3	J1	J6	J5	J4	J2
J2	J5	J1	J3	J6	J4
J6	J2	J4	J1	J5	J3

Figure 3-7 Sample of a Population of Chromosomes

3.8 Fitness Function

The fitness function is a measure of the quality of the schedule. The quality is defined in terms of the objective. The objectives of a user and the resource provider may differ.

Thus the user may have the following goals:

- Minimization of turn-around time
- Satisfying the deadline for the job, submitted by the user.

A resource provider may have the following goals:

- Minimization of the session completion time, also known as the makespan.
- Minimization of the idle time of the compute-nodes.

A resource allocation system has to try to schedule a number of jobs together during a processing session. Multiple jobs lead to a better utilization of the compute-nodes, since the scheduler may be able to allocate the idle gap times of compute-nodes, left out due to precedence constraints of one job, to the tasks of other jobs. Hence the fitness function represents cumulatively the objectives of a number of users.

Since the jobs can be of unequal size, for obtaining a cumulative result of multiple jobs and for obtaining a measure of multiple objectives, the raw measured values have to be normalized. One characteristic of a job, which helps in normalization, is the critical time of a job. This is the minimum time in which a job can be processed, if all the required resources are made available, without any delay.

In general the two objectives of the user and the two objectives of the resource provider are the four components of the fitness function.

3.9 The First Component: Turn-around time

A session can have "p" independent jobs. Let $t_{jobend}(i)$ be the turn-around time of the i-th job and let T_{je} be the cumulative value of the turn-around time for the p jobs. Let t_{st} be the end time of the last task for a given schedule.

$$T_{je} = \sum_{i=1}^{p} t_{jobend}(i) \qquad \text{..... (3.1)}$$

The normalized value of the cumulative turn-around time is as follows:

$$T_{jen} = T_{je} / (t_{st} * p) \qquad \text{..... (3.2)}$$

3.10 The Second Component: Deadline time

Let $t_{jobdline}(i)$ be the deadline time of the i-th job and let $T_{dt}(i)$ be the delay in meeting the deadline time requirement for the i-th job.

Deadline-Time: Algorithm

```
BEGIN
        Calculate Job_end_time for each job
        Obtain deadline_time given by the user for each job
        Total_deadline_time = 0;
        for all the jobs deadline time is calculated
          If (Job_end_time  > deadline_time_given) /* deadline has not been met */
                  Total_deadline_time = ( Job_end_time – deadline_time_given)
         end for
END
```

Initialize $T_{dt}(i)$ to 0 for i from 1 to p.

If $(t_{jobend}(i) > t_{jobdline}(i))$,

$$T_{dt}(i) = (t_{jobend}(i) - t_{jobdline}(i)) \quad \text{..... (3.3)}$$

$$T_{dt} = \sum_{i=1}^{p} (T_{dt}(i)) \quad \text{..... (3.4)}$$

Let T_{dt} and T_{dn} be the cumulative deadline delay and the normalized cumulative deadline delay respectively. Let t_{St} be the end time of the last task for a given schedule.

$$T_{dn} = T_{dt} / (t_{st} * p) \quad \text{..... (3.5)}$$

3.11 The Third Component: Makespan

Let $t_c(i)$ be the critical time of the i-th job and let Tc be the Max $(t_c(i))$ over all the jobs, being scheduled. Let t_{St} be the end time of the last task for a given schedule.

If ω be the makespan for the session,

$$\omega = 1 - (T_c / t_{st}) \quad \text{.....(3.6)}$$

3.12 The Fourth Component: Node Gap Time

This is a measure of the idle time of a node. The idle time arises due to the unutilized gaps, which have to be left to satisfy the precedence constraints of tasks of the jobs.

Let 'm' be the number of gaps in the j-th node and let 'n' be the number of compute-nodes. Let $t_{gs}(j, k)$ be the start time of the k-th gap of the j-th node and let $t_{ge}(j, k)$ be the end time of the k-th gap of the j-th node.

$$T_{ge} = \sum_{p=1}^{n} \left(\sum_{k=1}^{m} (t_{ge}(j,k) - t_{gs}(j,k)) \right) \quad \text{.....(3.7)}$$

If T_{gen} be the normalized cumulative idle time of all the n nodes,

$$T_{gen} = T_{ge} / (t_{st} * p) \quad \text{.....(3.8)}$$

3.13 The Fitness Function

The four components can be given weights of α, β, θ and λ, to obtain the fitness function F as follows:

$$F = 1 - ((\alpha * \omega + \beta * T_{jen} + \theta * T_{dn} + \lambda * T_{gen}) \ / (\alpha + \beta + \theta + \lambda))$$

.....(3.9)

The four weights (α,β,θ,λ) are used to prioritize any particular component in accordance with the needs of the jobs or the users or the resource provider.

3.14 Generation of a New Population

After the fitness function for each of the chromosomes of the initial population has been calculated, the average value of the fitness function and the standard deviation are computed. If the standard deviation (SD) should reach a sufficiently low threshold and if an acceptable solution should be available, the process of genetic algorithm is said to converge. Once the algorithm converges to a solution, the generation of a new population is not required. However since the initial population has been generated at random, without any consideration of the quality of the solution, SD is not likely to be below the threshold and the chromosomes available at this stage are not likely to provide an acceptable solution. Hence the scheduler has been designed to move immediately to the generation of the first new population, using the initial population as the basis. (The initial population is called as the existing population in the next set of paragraphs.)

Selection Methodology

To generate the new population, the method of elitism is used. First a roulette wheel of fitness functions can be constructed. Random numbers can be generated to select, out of the existing population, candidates for generating the new population. Using the selection method, crossover, mutation and elitism can be used jointly to generate the new population.

3.15 Crossover Process

A crossover rate (r_c) is chosen such that convergence may occur at a fast rate. This rate has been chosen for use in grid schedulers, through a large number of experiments. If the size of the existing population be P, rc * P parents are chosen, through the selection process, to create an equal number of children for the new population.

The crossover process can produce illegal chromosome, which can contain the duplicates of the same job(s). We use the described in [Goldberg, 1989], when we generate two child chromosomes from two parent chromosomes. It has two main steps. In the first step, we randomly generate a binary mask (M) of length l and a random integer j=0 or 1. Then at each position (i) of M such that M [i] = j (resp. M [i] = 1-j) we copy P_1 [i] to C_1 [i] (resp. P_2 [i] to C_2 [i]) [Ngom, 1998]. Figure 3-8 shows an example of the crossover process.

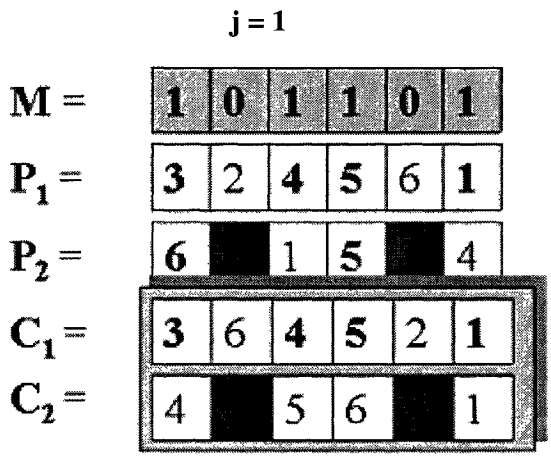

Figure 3-8 Crossover Process

3.16 Mutation Process

A mutation rate (r_m) is chosen such that convergence may occur at a fast rate. This rate has been chosen for use in grid schedulers, through a large number of experiments. If the size of the existing population be P, rm * P chromosomes are chosen, through the selection process, to create an equal number of chromosomes for the new population.

The mutation process is used to steer the new population out of a local optimum value. A high value of (r_m) can reverse the progress towards convergence. Hence this value has to be selected in each case through a careful study.

For mutation process we use randomly generate an integer j for each position (i) and then swap the elements at positions i and j in the permutation, as shown in the figure 3-9.

P =	2	4	**1**	5	**6**	3
C =	2	4	**6**	5	**1**	3

Figure 3-9 Mutation Process

3.17 New Chromosome through Elitism

To make the new population equal in number to the existing population, after the processes of crossover and mutation have been used, the remaining chromosomes are selected out of the existing population by making them identical with the best existing chromosomes, chosen through the selection process.

3.18 The GA-based Grid Scheduling Algorithm

```
BEGIN
    Generate initial population of λ individuals
    Evaluate individuals according to fitness function
        do until fitness value of all individual in the initial population is obtained.
            for each individual
                assign jobs-sub tasks to the given number of resources such that the earliest
                completion time with minimum idle time of resources can be obtained.
                Calculate the fitness function
        end do
    Select the best individuals from the initial population using Roulette wheel method.
      do until end criteria is not reached
        Generate a new population
                perform crossover operation
                perform mutation operation
                select remaining individual from previous population through policy of elitism
          Evaluate individuals according to fitness function
                do until fitness value of all individual in the new population is obtained.
                    for each individual
                        assign jobs-sub tasks to the given number of resources such that
                        the earliest completion time with minimum idle time of resources
                        can be obtained.
                        Calculate the fitness function
                end do
      end do
    END
```

3.19 Conclusion

This chapter describes the design of all the components of the scheduling algorithm developed in this thesis work. The next chapter will describe the testing process for validation and the results obtained when the validated scheduler was used. Many parts of the algorithm were adjusted after experiments and the tests were again performed to validate the module under test. The description of the design and the testing of the scheduler represent the final results.

Chapter 4

4 Experimental Results and Discussions

4.1 **Introduction**

The Genetic Algorithm based scheduler in this thesis has been designed so that it can schedule multiple jobs, with each job having multiple tasks. The tasks may have arbitrary precedence constraints and arbitrary execution times. A Directed Acyclic Graph can represent such a job.

The algorithm attempts multi-objective optimization for competing criterion. When the goals are such that when we attempt to improve one performance criterion, the other performance criterion may be degraded, the GA-based scheduler attempts to obtain a near optimum solution, while trying to satisfy all the objectives.

The first sets of tests obtain the results of scheduling single jobs with 50 to 3000 tasks with arbitrary precedence constraints and arbitrary execution times. Even though the jobs are NP hard [Kasahara, 2002], for specific number of compute-nodes, Kasahara has computed the optimum values of time, obtained through long computations on powerful computers. We compare the solutions obtained through our schedulers with the optimum results. We use our scheduler to run multiple STG jobs. From these experiments we obtain the best values of the crossover ratio, the mutation ratio and the convergence criterion.

The new version of Gridsim released on 12 November 2004, provides the possibility of space sharing for NIMROD/G scheduler. We select a problem that can be solved by the NIMROD/G scheduler on Gridsim. By solving the same problem by the NIMROD/G scheduler and by the scheduler developed in this thesis, the results can be compared.

After the three level validations, we schedule multiple STG jobs on a variable number of nodes and show that the scheduler is able to satisfy the multiple objectives and the solution converges in a reasonable number of generations.

4.2 The Complete Simulation Process

The complete simulation process, for developing the grid scheduler, requires meta data about the kind of jobs that may be expected in the most general case on the grid. After acquiring this information and the information about the available resources from a file, the simulation uses the Genetic Algorithm to converge to a solution, with the best value of the fitness function. The chromosome obtained through the process represents the best schedule. The schedule is used to map the tasks of each job to the available resources. Thereafter the values of the turn-around time, of the delay, if any, in meeting the deadline, of the makespan and of the idle time of the resources can be computed.

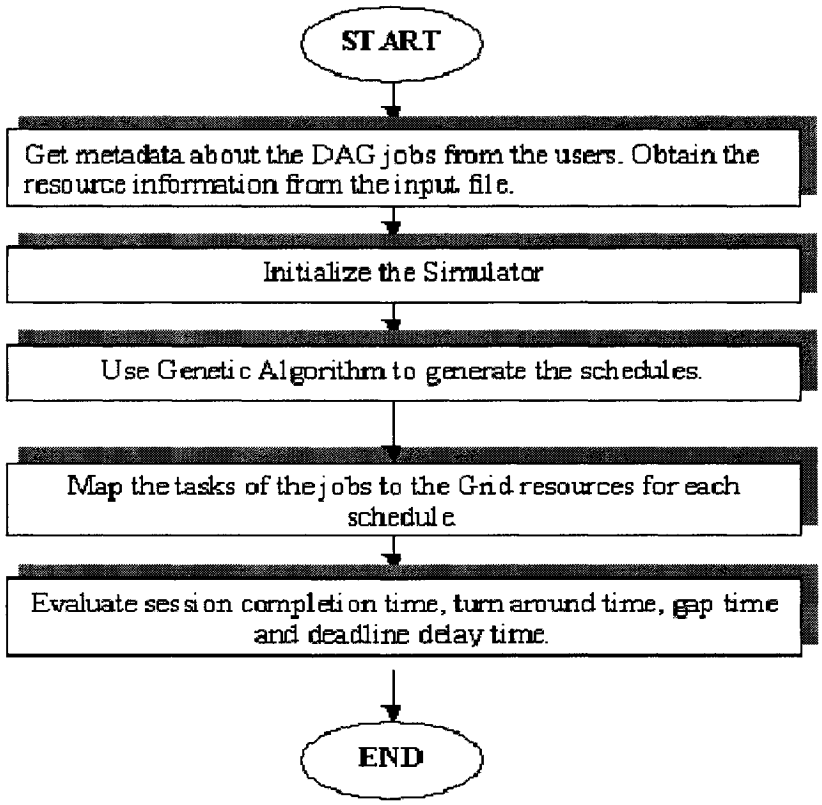

Figure 4-1 Process Flow of Simulation

After the evaluation of the fitness function, the convergence criterion is tested. If it is not satisfied, the last three steps are repeated till the process converges to a solution.

4.3 Test Environment for Genetic Algorithm-based Grid Scheduler

A GUI has been designed and implemented for conducting the tests. Figure 4-2 shows a screen shot of the user interface.

Submission of Jobs:

To enter the jobs, as a first step, the users have to enter the number of jobs that they want to use in the text field against the label "Total Jobs". Then the meta-data for each job has to be entered in the standard STG format. For each job, the total number of tasks has to be specified. For each task, the task #, its execution time and the task numbers of its parents have to be specified. For each job, a text file, containing the data for all its tasks, has to be prepared in the standard format. For the STG jobs, the files are available from [Kasahara, 2002]. To enter the files, enter the file name, the critical path length for the job and the deadline in the respective text fields and click the "Add" button. The filename of the job, the CP length and the Deadline time will appear in a single row in the text area of List of Jobs, located in the upper left part of the GUI. The same process is repeated for each of the jobs. At the end, all the jobs will be seen in the text area. The "Remove" button can be used to remove a job from the list.

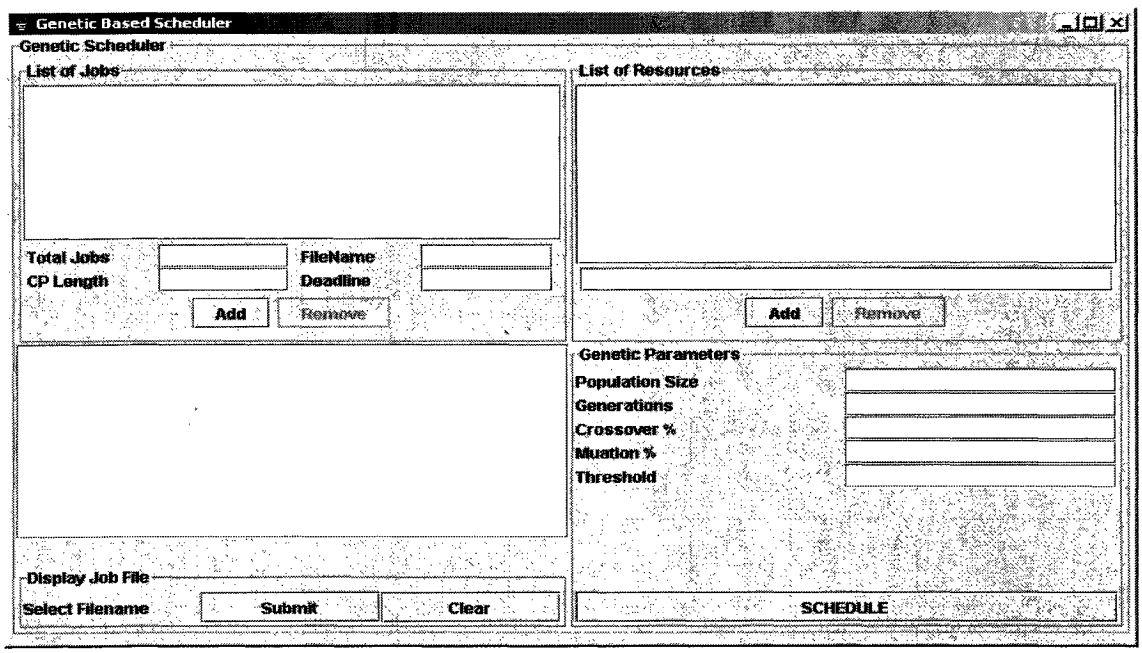

Figure 4-2 Graphical User Interface

On selecting a job in the text area of List of Jobs and on clicking the "Submit" button in the lower left part of GUI, the job file can be viewed in the text area of Display Job File, located in the lower left part of the GUI. On clicking the Clear button, the job file will be closed.

Entry of Resources:

The resources can be entered, one by one, through the text field and the "Add" button in the upper right part of the GUI. As a resource is entered, its name will appear in a single row in the text area of List of Resources, located in the upper right part of the GUI. The "Remove" button can be used to remove a resource from the list.

Entry of Genetic Parameters:

The size of the population of chromosomes, the maximum number of generations, the crossover ratio, the mutation ratio and the convergence value of the standard deviation are to be entered in the respective text fields.

With the above data, the scheduler is ready to generate the best schedule, which satisfies all the objectives in the best way, by providing the highest value of the fitness function. To obtain the solution, click on the "Schedule" button.

4.4 Tests with Standard DAG jobs

Kasahara and Tobita [Kasahara, 2002] have diligently obtained the optimum schedule length (or its range) for mapping DAGs to a specific number of nodes. The DAGs have been worked out by using random selection method to select both the precedence constraints and the execution times for tasks in a job. DAGs with 50 to 3000 tasks are available on the web-site [http://www.kasahara.elec.waseda.ac.jp/schedule/index.html]. Each DAG has been mapped to 2 to 16 nodes. For each case the optimum schedule length has been computed by Kasahara. through the use of high computational power. In practice even for batch mode, a scheduler cannot use a similar processing power for obtaining the optimum value. Hence the optimum value (or its range) specified in the STG set is considered to be the best that a scheduler can ever achieve in practice.

In this work, we have compared the values achieved through our algorithm, with the optimum values given in the STG set. This validates our algorithm.

Eight STG jobs with 50 to 3000 tasks were selected for the testing. The tasks in each job have arbitrary precedence constraints in that every task may have, in general, multiple parents and multiple children. The tasks have multiple layers and each layer has arbitrary width. Moreover the execution time of each task varies arbitrarily. The experiments on our schedulers have been conducted with 2 to 16 nodes. For the STG jobs, [Kasahara, 2002] have provided the optimum value of schedule length. The schedule length is equal to the makespan, when a single job is run on the scheduler. So in this experiment, the values of makespan are obtained. These are compared with the optimum values available from [Kasahara, 2002].

			No. of Nodes			
S.No.	# of Tasks		2	4	8	16
1	50	OP	149	94	89	89
		OB	153	106	89	89
		%age Error	2.68	12.76		
2	100	OP	267	134	81	81
		OB	269	137	88	81
		%age Error	0.7	2.2	8.6	
3	300	OP	860	436	383	383
		OB	866	465	385	383
		%age Error	0.69	6.6	0.52	
4	500	OP	1364	737	737	737
		OB	1371	803	738	737
		%age Error	0.51	8.9	0.13	
5	750	OP	2060	1030	519	512
		OB	2062	1042	567	512
		%age Error	0.09	1.1	9.24	
6	1000	OP	2721	1361	1079	1079
		OB	2731	1413	1083	1079
		%age Error	0.36	3.82	0.37	
7	2000	OP	5576	2954	2821	2821
		OB	5647	3191	2824	2821
		%age Error	1.27	8.02	0.106	
8	3000	OP	8290	4145	2848	2848
		OB	8313	4267	2899	2848
		%age Error	2.77	2.94	1.79	

OP: The optimum value [Kasahara]
OB: The measured makespan obtained by our Algorithm

Table 2: Measured Makespan vs Optimum Value for STG Jobs with 50 to 3000 Tasks

Table 2 shows the values of makespan, obtained by these experiments on our scheduler for the cases of 50 to 3000 tasks and with 2 to 16 nodes. It also shows the optimum value for each case [Kasahara, 2002]. For lower number of nodes, the optimization effort required is rather high. Therefore in general, a heuristic algorithm will work better for larger number of nodes, than it does for smaller number of nodes. The case of 2 nodes may be considered as a roughly degenerate case. When the choice is limited to only two nodes, the optimization process cannot do much better than the heuristic algorithm. Figure 4-3 shows the results graphically.

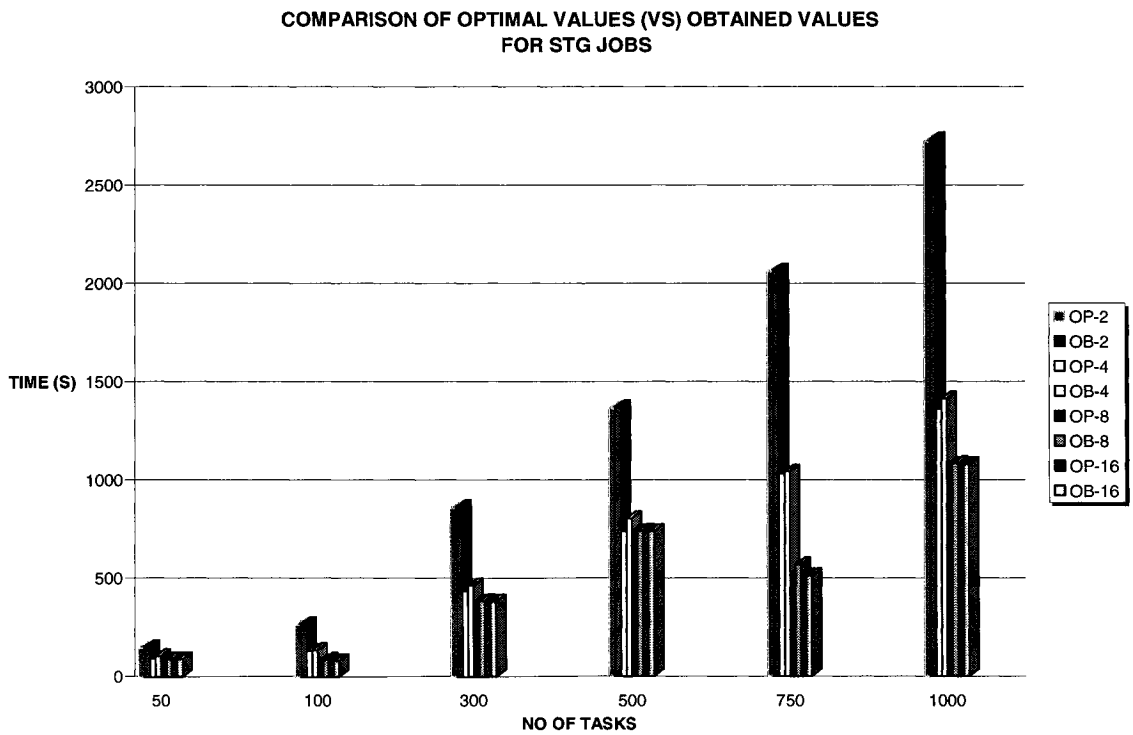

Figure 4-3 The measured Makespan versus the Optimum Values

4.5 Tuning the Parameters of the Genetic Algorithm

There are two crucial parameters, which have to be selected for the application at hand. These are the crossover rate and the mutation rate. To adjust the values, we conducted experiments to select appropriate values. Ten STG jobs with 50 tasks each were selected. An initial population of 100 was generated. A large set of values for crossover rate and mutation rate was used. In

each case, new generations of population were created till convergence was reached with standard deviation of 6.3×10^{-7}.

For mutation ratio, experiments showed that a value of 0.01 would only lead to convergence. With any higher value, and even with a large set of crossover ratios, convergence could not be reached. Hence the mutation ratio was set at 0.01.

With this value of mutation ratio, experiments for crossover ratio of 0.7 to 0.8 are shown in Figure 4-4. It is found that with crossover ratio of 0.74 and 0.76, the same value of fitness function was obtained.

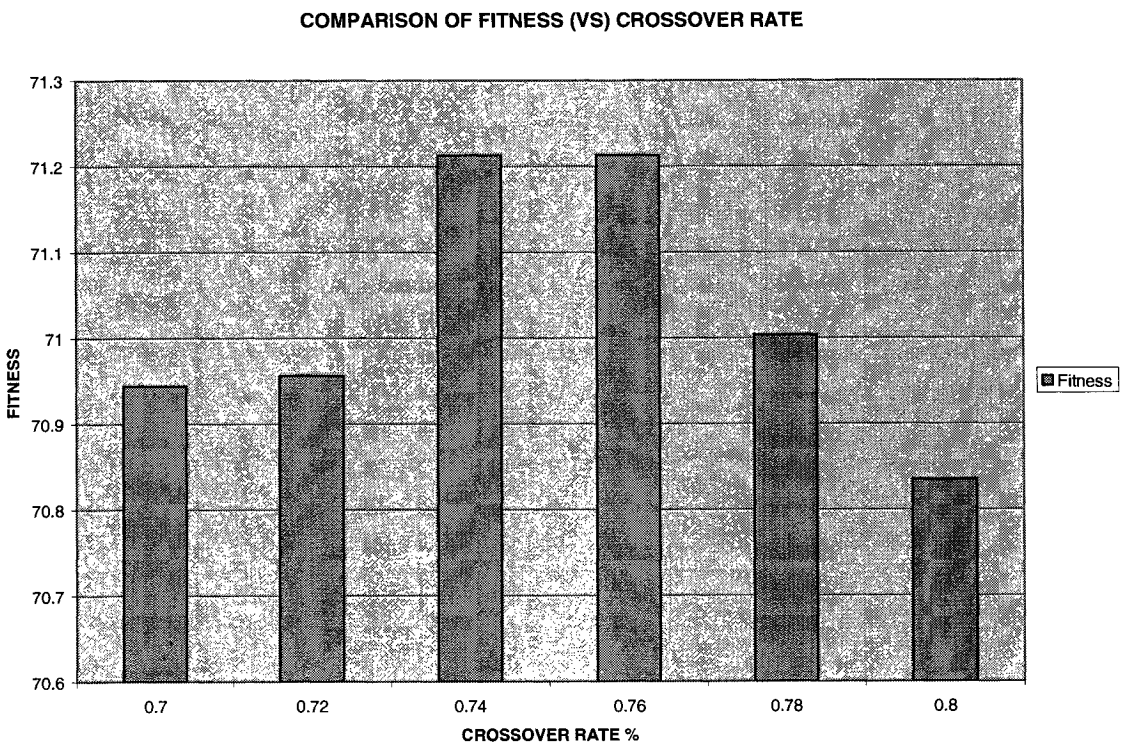

Figure 4-4 Fitness versus the Crossover Ratio

To select the number of generations and the size of population, required for getting the schedule, experiments were conducted with a change in population size from 20 to 150. Figures 4-4 and Figure 4-5 show the results.

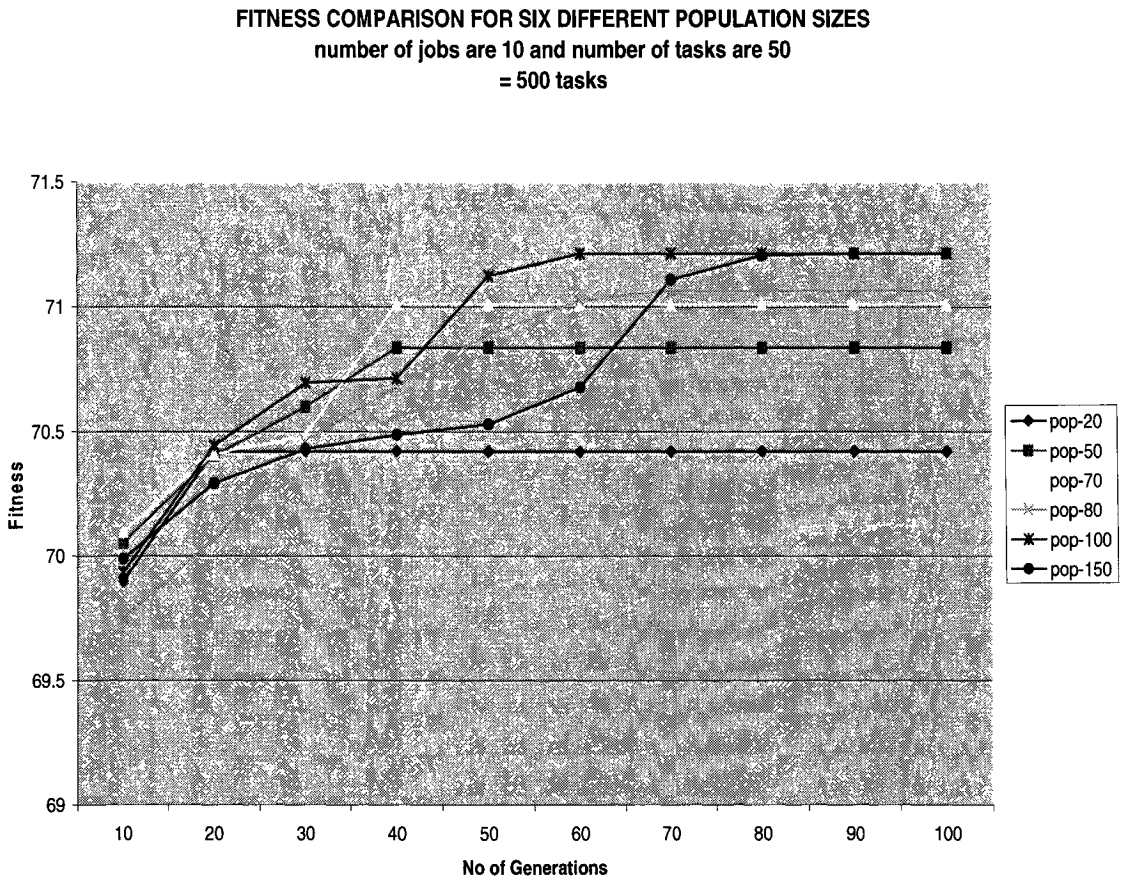

Figure 4-5 Fitness versus # of Generations

Figure 4-5 is a graph of Fitness versus Number of Generations required for convergence. The graph shows that a population size of 100 or 150 is able to give the same result after 80 generations or more.

Figure 4-6 shows a plot of standard deviation versus the number of generations for convergence. Population sizes of 20 to 150 are taken. It shows that as the size of population is increased, the convergence requires a larger number of generations. Thus Figure 4-5 shows that a population of 100 should converge in 80 generations.

For all other experiments for this work, we have taken the standard values of crossover ratio of 0.76, mutation ratio of 0.01 and a population size of 100.

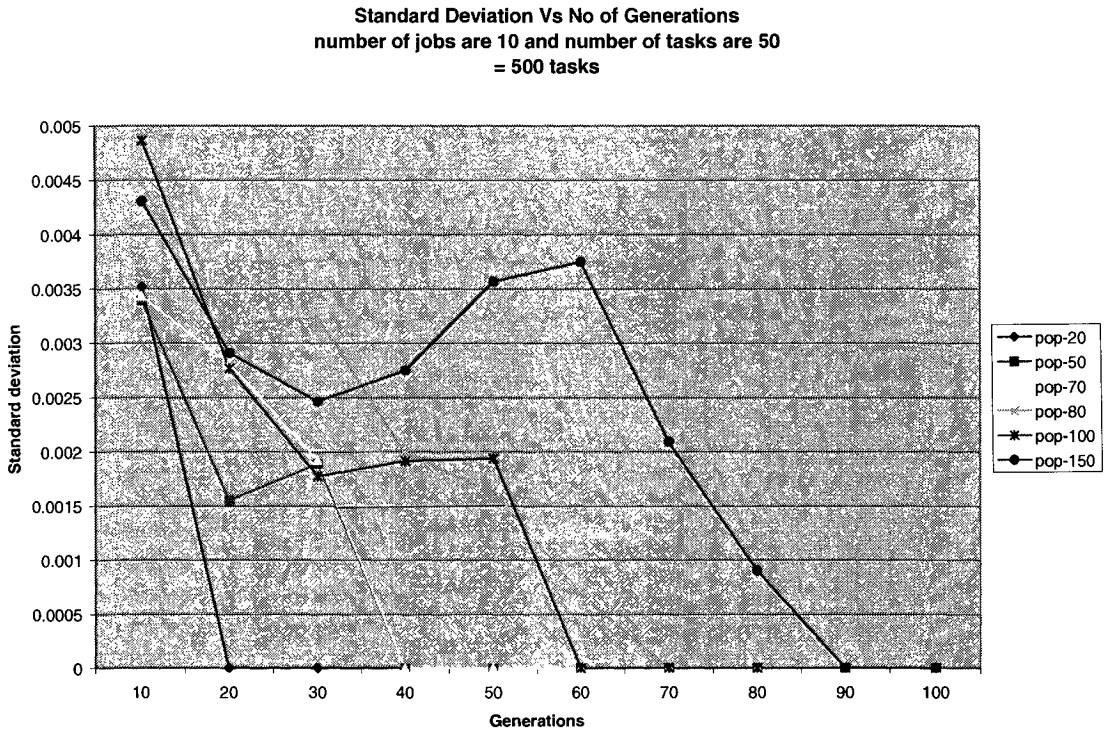

Figure 4-6 Standard Deviation versus # of Generations

With these parameters, another experiment for scheduling ten different jobs by using the GA-based scheduler was conducted. Each of the 10 jobs was a different job. The number of inter-dependent tasks in each job varied from 50 to 1000. The total number of tasks in the ten jobs was 3650. Using these complex sets of jobs, the values of fitness function at 10 to 100 generations are plotted in Figure 4-7.

The Figure again shows that Fitness varies very little above 80 generations and it becomes nearly stable at 100 generations.

We can draw two conclusions from the experiment. The parameters chosen for the crossover ratio and mutation ratio seem to be appropriate. Secondly a population size of 100 seems to work for the system.

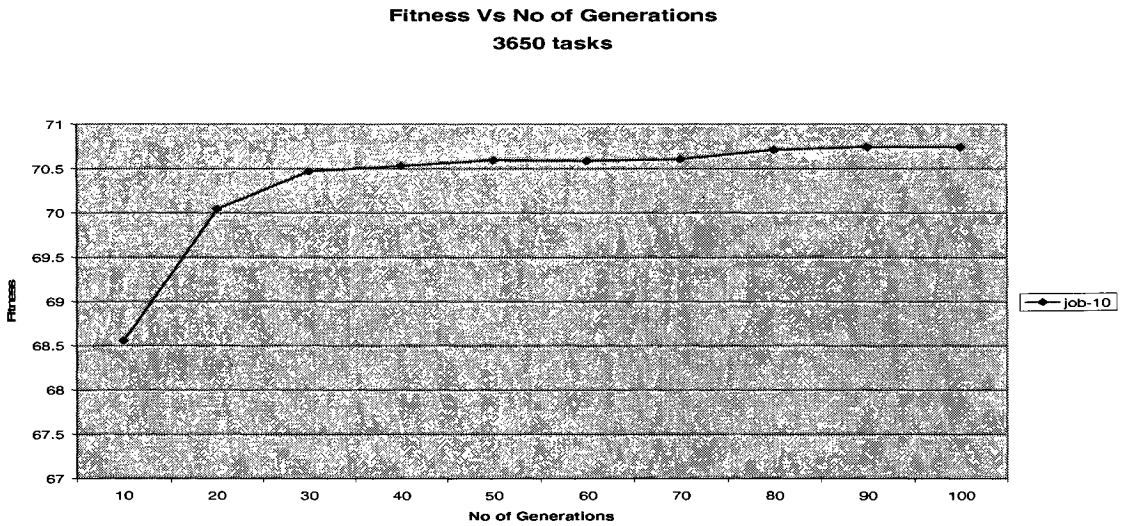

Figure 4-7 Fitness versus # of Generations for 10 Jobs of 3650 Tasks

Figure 4-8 shows Fitness versus the number of generations for 35 STG jobs with 10,750 tasks. The graph again shows that after about 80 generations, the value of fitness stabilizes. So we choose, for our scheduler, a default value of 100 generations.

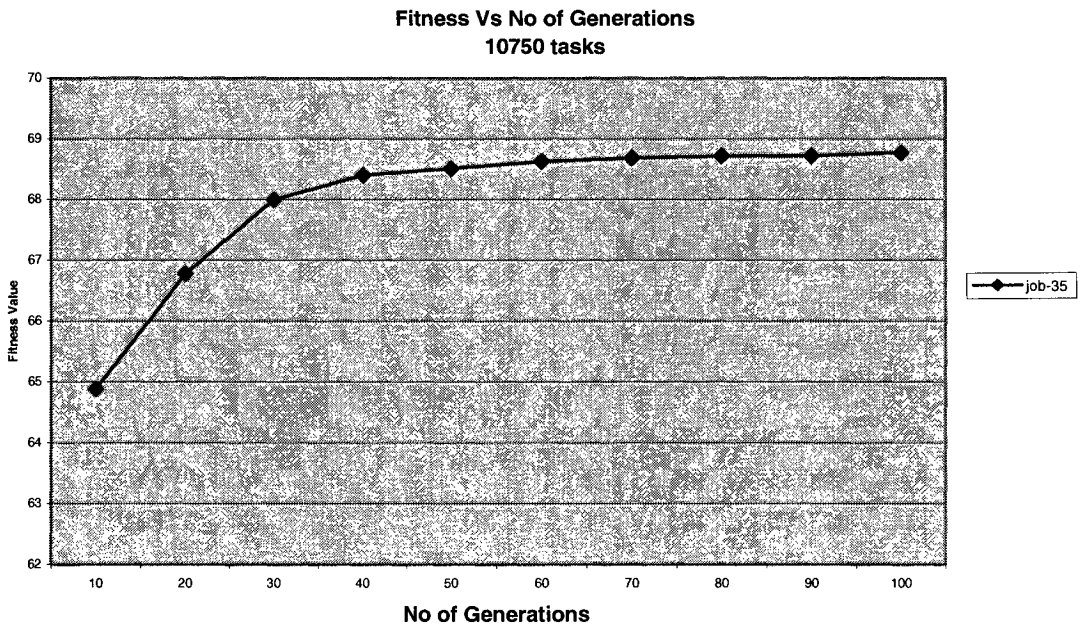

Figure 4-8 Fitness versus # of Generations for 35 Jobs of 10750 Tasks

4.6 Comparison with NIMROD/G scheduler through Gridsim

Every scheduler uses architecture, which is unique. Secondly no other scheduler with multiple jobs, with inter-dependencies among tasks, and with multiple objectives is available. Moreover comparison with another scheduler is possible only with the kind of problems that the other scheduler can manage.

The same group of researchers, who have designed Nimrod-G, has designed Gridsim simulator. It is able to accept independent jobs with space sharing. However the jobs in Nimrod-G cannot accept any sub-jobs, called tasks. The objective of the Nimrod-G scheduler is to optimize the cost and deadline time. For simulation, the strategy of the broker is set to "time-optimization".

A set of independent jobs is scheduled using Nimrod-G scheduler on the Gridsim simulator. The simulator can give the time used for the experimentation. The same set of jobs is tried on our scheduler.

A set of 30 different jobs is considered. The jobs are mapped onto a 2 to 8 machine system. The resultant values of makespan, obtained for both Nimrod-G and our GA based scheduler, are shown in Table 4. The results show clearly that in every case the GA based scheduler is able to obtain better results than Nimrod-G results obtained from the Gridsim simulation. This is due to the use of FCFS policy or its variants by Nimrod-G. "A resource with multiple machines is treated as a distributed memory cluster and is managed as a space-shared system using FCFS scheduling policy or its variants." [Buyya, 2002b]. The GA-based scheduler considers all the jobs that it is scheduling, as a single set and attempts to find a schedule, which satisfies the objective. Nimrod-G first schedules the job, which is entered first, even though it is scheduling 10 (or 20 or 30) jobs at the same time. This leads to a worse performance by Nimrod-G.

10 jobs		Machine	Machine	Machine	Machine	Machine	Machine	Machine
		2	3	4	5	6	7	8
	Nimrod-G	92	72	67	52	52	52	52
	GA (Avg)	90	60	50	40	40	40	40

20 jobs		Machine	Machine	Machine	Machine	Machine	Machine	Machine
		2	3	4	5	6	7	8
	Nimrod-G	617	427	317	262	237	207	198
	GA (Avg)	575	290	290	230	195	165	145

30 jobs		Machine	Machine	Machine	Machine	Machine	Machine	Machine
		2	3	4	5	6	7	8
	Nimrod-G	39702	27052	23502	19052	15952	13653	13603
	GA (Avg)	36800	24600	18400	14800	12350	10500	9500

Table 3 Makespan Values Obtained through Nimrod-G-Gridsim Simulator and GA based Scheduler

Figure 4-9 shows the comparison of Nimrod-G and GA-based Scheduler graphically.

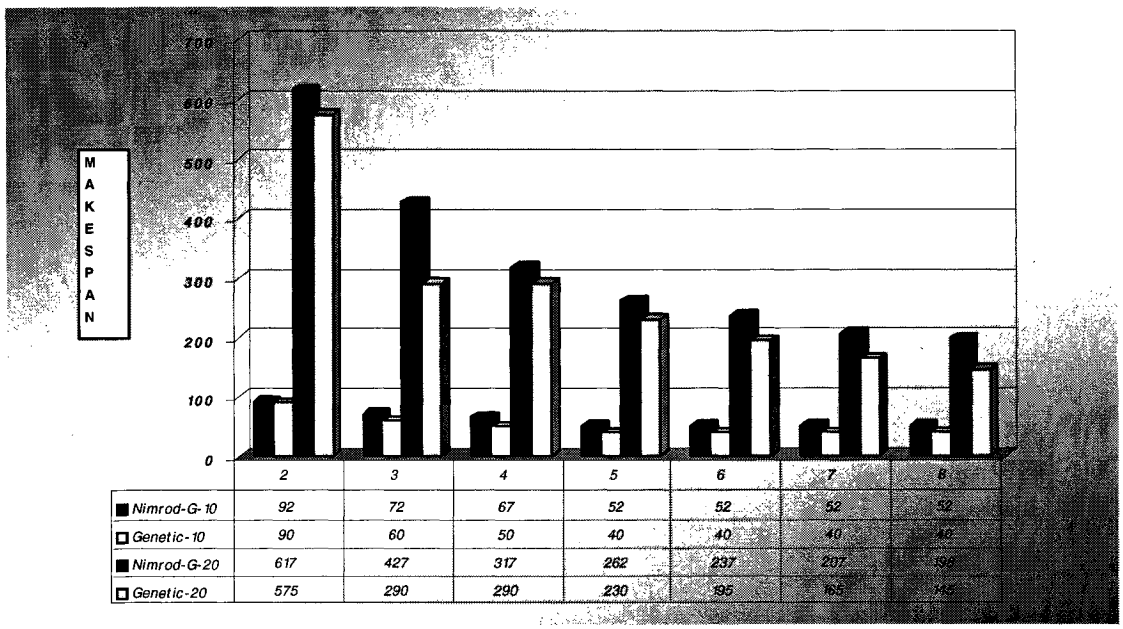

Figure 4-9 Comparison of Nimrod-G and GA – based Scheduler

4.7 Conclusion

For the case of scheduling multiple jobs, with arbitrary inter-dependencies among the tasks of each job, on a distributed system of heterogeneous compute resources, a theoretical optimum value of session completion time is not available in the literature. In section 4.4 we present in our work the scheduled length that our scheduler is able to achieve for a set of multiple STG jobs and a specified goal. This set of results can be considered as an addition to the STG set.

Chapter 5

5 Conclusion and Future Work

High-speed data networks today girdle the globe. These have made it technically possible for applications to be processed through computational resources, spread out over large geographical areas. A large number of compute resources are used intermittently during the working part of the day only. Thus we have the hardware and network resources to be able to use compute-resources more effectively through a grid. But we require the middleware and the creativity to build an architecture that can use these resources. This thesis makes a contribution to the design of a working grid by designing a scheduler, based on a hierarchical architectural model. The algorithm of the scheduler has been tested exhaustively and compared with all the results, available for earlier work.

We have designed a Genetic Algorithm based scheduler for the architecture discussed in Chapter 3. It is able to schedule multiple jobs with arbitrary precedence constraints and arbitrary execution times, while satisfying multiple objectives. We have verified its functionally for as many as 10, 750 tasks, which may be much more than what a scheduler may be called upon to do at a time. We have also worked out the crossover ratio, the mutation ratio and the size of population that may be appropriate for the scheduler. We have compared with the only other scheduler, Nimrod/G, for which a simulator is available and we find that our algorithm is able to work much better. The paper states [Spooner, 2003a] that the TITAN system is under development and is, at present, able to handle relatively simple jobs only. The example shown in the paper considers jobs with independent sub tasks, which have no communication or precedence constraints with one another. We have taken multiple jobs, which are realistic and worked out the schedules. We have developed a Fitness function, which is appropriate for grid scheduling. Our results for the session completion time for multiple STG jobs may be considered as an addition to the standard optimum time, available for single STG jobs. Moreover the method developed by us can be used to develop a large database of standard optimum session completion time and minimum idle time of compute nodes for sets of STG jobs and for specified number of compute-nodes. Such a database may prove to be of use to researchers for comparing the characteristics of new scheduling algorithms.

Future Work

The scheduler has been tested exhaustively. But if the resource allocation system to be handled should be big, the scheduler may have to handle a much larger number of jobs. In such a case for faster convergence, a larger computational power may be required. This can be easily obtained from the grid, if the scheduler can be modified to run on a parallel set of machines using parallel Genetic Algorithm.

Another open problem is to find a heuristic for efficient optimization of the second level of dependent jobs of the schedules obtained through Genetic Algorithm.

One requirement in the grids is for resource reservation. This requirement has to be built into a Resource Allocation System by ensuring that the resource data provided to the scheduler takes reservations into account. After building such systems, the scheduler will also have to be tuned to be able to accept this facility.

The scheduler will have to be integrated with other components, which are being studied by our research group. The whole system can be made compatible with the Globus tool-kit, which is emerging as the standard. An integrated system, including the scheduler, may prove to be of great use to the entire grid community.

Bibliography

[Abawajy, 2003] J. H. Abawajy, S.P. Dandamudi. Parallel Job Scheduling on Multi-Cluster Computing Systems. Proceedings of IEEE International Conference on Cluster Computing (Cluster 2003), pp.11-18, 2003.

[Abramson, 2000] D. Abramson, J. Giddy, et al, High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?, In Proceedings of the 14th International Conference on Parallel and Distributed Processing Symposium (IPDPS-00), pp.520–528, 2000.

[Berman, 2003] Francine Berman, Richard Wolski, Henri Casanova, et al, Adaptive Computing on the Grid Using AppLes, IEEE Transactions On Parallel and Distributed Systems, v.14, pp.369-382, 2003.

[Berman, 2001] Francine Berman, Andrew Chien, Keith Cooper, et al, The GrADS Project: Software support for high-level Grid application development, The International Journal of High Performance Computing Applications, v.15 n.4, pp.327–344, 2001.

[Buyya, 2002a] Rajkumar Buyya, Muthucumaru Maheswaran, et al, A taxonomy and survey of grid resource management systems for distributed computing, The Journal of Software Practice and Experience, v.32 n.2, pp.135–164, 2002.

[Buyya, 2002b] Rajkumar Buyya and Manzur Murshed, GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing, Concurrency and Computation: Practice and Experience, *Concurrency Computat.: Pract. Exper.* 2002, v.14, pp.1175–1220.

[Buyya, 2002c] Rajkumar Buyya, Economic-based Distributed Resource Management and Scheduling for Grid Computing, Ph.D. Thesis, Monash University Australia, April 2002.

[Buyya, 2000a] R. Buyya, D. Abramson, J. Giddy, Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid, Proceeding of the HPC ASIA'2000, the 4th International Conference on High Performance Computing in Asia-Pacific Region, Beijing, China, IEEE Computer Society Press, USA, 2000

[Buyya, 2000b] Rajkumar Buyya, Baikunth Nath, et al, Nature's Heuristics for Scheduling Jobs on Computational Grids, The 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000), 2000, India.

[Casavant, 1988] T.L. Casavant, J.G. Kuhl, A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems, IEEE Transaction on Software Engineering, v.14 n.2, pp.141-154, 1988.

[Casanova, 2000] H. Casanova, A. Legrand, D. Zagorodnov, et al, Heuristics for Scheduling Parameter Sweep Applications in Grid Environments, In Proceedings of the 9th Heterogeneous Computing Workshop (HCW'00), pp.349--363, 2000.

[Casanova, 2003] H. Casanova, H. Dail, F. Berman, A Decoupled Scheduling Approach for Grid Application Development Environments, Journal of Parallel and Distributed Computing (JPDC), Special issue on Grid Computing, v.63 n.5, pp.505-524, 2003.

[Czajkowski, 1997] K. Czajkowski, I. Foster, et al, A resource management architecture for metacomputing systems, The 4th Workshop on Job Scheduling Strategies for Parallel Processing, pp.62–82, Springer-Verlag LNCS, 1997.

[Foster, 1999] Foster, C. Kesselman, et al, A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation, International Workshop on Quality of Service, 1999.

[Foster, 1998] Foster, C. Kesselman, et al, Computational Grids, The Grid: Blueprint for a New Computing Infrastructure, Morgan-Kaufman, San Fransisco, 1998.

[Foster, 1997a] S. Fitzgerald, I. Foster, et al, A directory service for configuring high-performance distributed computations, Sixth IEEE Symposium On High Performance Distributed Computing, pp. 365-375, 1997.

[Foster, 1997b] Foster, C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit, International Journal of Supercomputer Applications, v.11 n.2, pp.115-128, 1997

[Frey, 2002] James Frey, Todd Tannenbaum, et al, Condor-G: A Computation Management Agent for Multi-Institutional Grids, Journal of Cluster Computing, v.5, pp.237-246, 2002.

[Goldberg, 1989] Goldberg D.E, Genetic Algorithm in Search, Optimization, and Machine Learning, Addison-Wesley, Reading, MA.

[Holland, 2003] Matthias Hovestadt, Odej Kao, et al, Scheduling in HPC Resource Management Systems: Queuing vs. Planning, Proc. of the 9th International Workshop on Job Scheduling Strategies for Parallel Processing, volume 2862 of Lecture Notes in Computer Science, pp.1–20. Springer, 2003.

[Jackson, 2001] D. Jackson, Q. Snell, et al, Core Algorithms of the Maui Scheduler, Proceedings of 7th Workshop on Job Scheduling Strategies for Parallel Processing, volume 2221 of Lecture Notes in Computer Science, pp. 87–103. Springer Verlag, 2001.

[Kasahara, 2002] H.Kasahara, T.Tobita, A standard task graph set for fair evaluation of multiprocessor scheduling algorithms. Journal of Scheduling, vol 5, pp.379-394, 2002

[Kruskal, 1984] C. Kruskal, A. Weiss, Allocating independent subtasks on parallel processors, IEEE Transactions on Software Engineering, 11, pp.1001-1016, 1984.

[Litzkow, 1988] M.Litzkow, M.Livny, Condor -A Hunter of Idle Workstations. Proc. of the 8th International Conference of Distributed Computing Systems, pp.104-111.University of Wisconsin, Madison, 1988.

[Liu, 2004] M.L.Liu, Distributed Computing Principles and Applications, Pearson Addison-Wesley, ISBN 0-201-79644-9, 2004.

[Ngom, 1998] A. Ngom, 1997, Genetic algorithms for the jump number scheduling problem, Order - A Journal on the Theory of Ordered Sets and its Applications,pp.59-73,1998.

[Maheswaran, 1999] M. Maheswaran, S. Ali, et al, Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems, In 8th Heterogeneous Computing Workshop (HCW'99), pp.30-45, 1999.

[Schwiegelshohn, 2002a] U. Schwiegelshohn, R. Yahyapour, et. al. On Advantages of Grid Computing for Parallel Job Scheduling. In Proceedings of the 2nd IEEE International Symposium on Cluster Computing and the Grid (CC-GRID 2002), pp. 39-46, 2002.

[Schwiegelshohn, 2000b] U. Schwiegelshohn, A. Streit, R. Yahyapour, et.al, Evaluation of job-scheduling strategies for grid computing, Proceedings of 1st IEEE/ACM International Workshop on Grid Computing (Grid 2000), v.1971, pp.191–202, 2000.

[Spooner, 2003a] D. P Spooner, SA Jarvis, et al, Local Grid Scheduling Techniques using Performance Prediction, IEE Proceedings - Computers and Digital Techniques, v.150 n.2, pp.87-96, 2003.

[Spooner, 2002b] D. P Spooner, SA Jarvis, et al, Localised Workload Management using Performance Prediction and QoS Contracts, Proc. 18th Annual UK Performance Engineering Workshop, pp. 69-80, 2002.

[Wolski, 2003a] Rich Wolski, Experiences with predicting resource performance on-line in computational grid settings, ACM SIGMETRICS Perform. Eval. Rev. v.30 n.4, pp.41-49, 2003, ACM Press.

[Wolski, 1999b] R. Wolski, N. T. Spring, et al. The Network Weather Service: A distributed resource performance forecasting service for metacomputing, The Journal of Future Generation Computing Systems, pp.1-19, 1999.

[Wright, 2001] D Wright, Cheap Cycles from the Desktop to the Dedicated Cluster: Combining Opportunistic and Dedicated Scheduling with Condor, Proceeding of HPC Revolution 01, Illinois, 2001.

[Web-500-list] http://www.top500.org

[Web-Sharcnet] http://www.sharcnet.ca

[Web-GlobalGrid] http://www.globagridforum.org

[Zhang, 2003] Yanyong Zhang, Hubertus Franke, et al, An Integrated Approach to Parallel Scheduling Using Gang-Scheduling, Backfilling, and Migration, In IEEE Transactions On Parallel and Distributed Systems, v. 14, pp.236-247, 2003.

Appendix

Grid System Taxonomy

A CLASSIFICATION BASED ON ADMINISTRATIVE DOMAINS

An **IntraGrid** may be controlled by a single, multi-location organization. Sharcnet is an example of such a grid. This consists of big clusters at 4 universities connected by a dedicated Internet.

An **Extra Grid** may couple two or more grids. ERP systems of today use such grids whereby a business entity may be connected to its suppliers and its sales organizations.

Global Grid is the kind of grid being conceptualized by the Global Grid Forum. Such a grid would have a large number of resource providers connected through Internet to the users. Middleware for such a grid is being developed by a number of research groups of GGF and at various Universities.

A CLASSIFICATION BASED ON TYPE OF APPLICATION

Grid systems can be classified, into three types [Buyya, 2002a].

- Computational grids
- Data grids
- Service grids

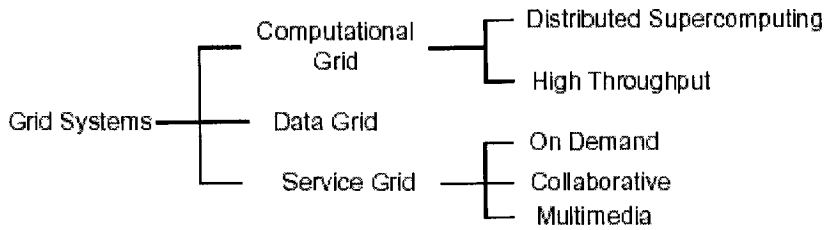

Figure 1 : Grid Systems Taxonomy (Buyya, 2002a, pp: 3)

The aggregate computational power of a computational grid is larger than the computational power of any single constituent of the grid. High Performance computing problems, which require large computational power, use the grids for parallel computing. [Buyya, 2002a].

A data grid provides large data repositories such that both the data storage as well as the users may be distributed over a large geographical area. Users may mine the data for studies in different ways. As an example, the astronomers, as a loose group, have been able to create a large data grid. Large telescope systems continuously scan the outer space and feed the data into large data repositories. These are then used by astronomers from around the world. [Buyya, 2002a].

Service grids provide services not available at a single site from a single machine. Examples of such grids are On Demand computing, Multimedia computing or Collaborative computing.

Vita Auctoris

NAME: Mona Aggarwal

COUNTRY OF BIRTH: India

YEAR OF BIRTH: 1976

EDUCATION: Bachelor of Science in Computer Science with Software Engineering Option (Co-op)
University of Windsor
Windsor, ON, Canada, 2001

Master of Science in Computer Science
University of Windsor
Windsor, ON, Canada, 2004