EFFICIENT SCHEDULING IN DISTRIBUTED COMPUTING ON GRID

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ÖZGÜR KAYA

IN PARTIAL FULFILMENT THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

NOVEMBER 2006

Approval of the Graduate School of Natural and Applied Sciences

_____

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science

_____

Prof. Dr. Ayşe Kiper
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science

_____

Prof. Dr. Müslim Bozyiğit
Supervisor

Examining Committee Members

Prof. Dr. F. Payidar Genç (METU,CENG)_____

Prof. Dr. Müslim Bozyiğit (METU,CENG)_____

Dr. Atilla Özgit (METU,CENG)_____

Dr. Ece (Güran) Schmidt (METU,EE) _____

Dr. Onur T. Şehitoğlu (METU,CENG)_____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : Özgür, Kaya

Signature :

ABSTRACT

SCHEDULING ALGORITHMS FOR GRID COMPUTING

Kaya, Özgür
M.S., Department of Computer Engineering
Supervisor: Prof. Dr. Müslim Bozyiğit
November 2006, 123 pages

Today many computing resources distributed geographically are idle much of time. The aim of the grid computing is collecting these resources into a single system. It helps to solve problems that are too complex for a single PC. The grid applications can be used in financial services, industry, government services, military, medicine, and many engineering applications. Scheduling plays a critical role in the efficient and effective management of resources to achieve high performance on grid computing environment. Due to the heterogeneity and highly dynamic nature of grid, developing scheduling algorithms for grid computing involves some challenges. In this work, we concentrate on efficient scheduling of distributed tasks on grid. We propose a novel scheduling heuristic for bag-of-tasks applications. The proposed algorithm primarily makes use of history based runtime estimation. The history stores information about the applications whose runtimes and other specific properties are recorded during the previous executions. Scheduling decisions are made according to similarity between the applications. Definition of similarity is an important aspect of this approach, apart from the best

resource allocation. The aim of this scheduling algorithm (HISA-History Injected Scheduling Algorithm) is to define and find the similarity, and assign the job to the most suitable resource, making use of the similarity. In our evaluation, we use Grid simulation tool called GridSim. A number of intensive experiments with various simulation settings have been conducted. Based on the experimental results, the effectiveness of HISA scheduling heuristic is studied and compared to the other scheduling algorithms embedded in GridSim. The results show that history injection improves the performance of future job submissions on a grid.

Keywords: Grid, Grid Computing, Scheduling, Grid Scheduling, History Based Scheduling, Static and Dynamic Scheduling.

ÖZ

GRİD HESAPLAMA İÇİN ZAMANLAMA ALGORİTMALARI

Kaya, Özgür
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Müslim Bozyiğit
Kasım 2006, 123 sayfa

Bugün, coğrafik olarak dağıtık halde bulunan hesaplama kaynakları, zamanlarının büyük bir kısmını boşa harcamaktadırlar. Grid'in amacı bu kaynakları tek bir sistem içerisinde toplamaktır. Bu durum, bir tek PC için çözümü çok zor olan veya çok vakit alan kompleks problemlerin çözümüne yardım eder. Grid uygulamaları finansal servislerde, endüstride, devlette, savunma alanlarında, tıpta, ve birçok mühendislik uygulamalarında kullanılabilir. Grid hesaplama ortamında, yüksek performansa ulaşmak için kaynakların verimli ve etkili bir biçimde kullanılmasında zaman kritik bir rol oynar. Grid'in yapısı son derece dinamik ve farklı olduğundan, Grid hesaplama için zamanlama algoritmalarının geliştirilmesi beraberinde uğraştırıcı zorluklar getirir. Bu çalışmada, Grid üzerinde dağıtık işlerin verimli zamanlaması üzerine yoğunlaştık. İş kümesi uygulamaları için yeni bir zamanlama algoritması amaçladık. Amaçlanan algoritma, işlerin geçmişteki işleyişlerini hesaba katmaktadır. Geçmiş, uygulamaların daha önceki çalışmaları boyunca kaydedilen çalışma-zamanları ve diğer özellikleri hakkındaki bilgileri

saklar. Zamanlama kararları bu uygulamalar arasındaki benzerliğe bakılarak alınır. Benzerliğin tanımlanması, en iyi kaynak ataması bir tarafa, bu yaklaşımda ortaya çıkan önemli bir niteliktir. Zamanlama algoritması HISA'nın amacı bu benzerliğin tanımlanması ve bulunması, daha sonra da bu benzerliği kullanarak işin atanacağı en uygun kaynağın bulunmasıdır. Bu değerlendirmemizde, GridSim diye bilinen bir Grid simulasyon aracı kullandık. Çeşitli simulasyon ayarları ile çok sayıda deneysel çalışma yaptık. Bu deneylerin ışığında, kullanılan zamanlama algoritmasının etkililiği değerlendirildi. Test sonuçları, HISA'nın, gelecek işlerin bir Grid'e atanmasındaki performansı geliştirdiğini göstermiştir.

Anahtar Kelimeler: Grid, Grid Hesaplama, Zamanlama, Zamanlama Algoritmaları, Geçmiş Temelli Zamanlama, Statik ve Dinamik Zamanlama

To My Family and my unborn child

ACKNOWLEDGMENTS

I wish to express my deepest gratitude to my supervisor, Prof. Dr. Müslim Bozyiğit for his guidance, advice, criticism, encouragements, suggestions and insight throughout this work. And I would also like to thank my family members especially my wife for their endless love and support. I would also like to mention my friend Yusuf Tambag and Seda Öztürk who were beside me whenever I looked around for help.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Today many computing resources distributed geographically are idle much of time. The aim of the grid computing is collecting these resources into a single system. It helps to solve problems that are too complex for a single PC. In Grid computing, the goal is to create the illusion of a simple yet large and powerful self managing virtual computer out of a large collection of connected heterogeneous systems sharing various combinations of resources. There are no practical tools for transforming arbitrary applications to exploit the parallel capabilities of a grid. There are some practical tools that skilled application designers can use to write a parallel grid application. However, transformation of applications can be a difficult task. In some cases, the burden of creating such environments might outpace the effort required to enhance the capabilities of such systems.

Scheduling plays a critical role in the efficient and effective management of resources to achieve required performance on grid computing environment. Due to the heterogeneity and highly dynamic nature of grids, developing scheduling algorithms for grid computing involves some further challenges**.**

In this study, we attempt to develop a Grid scheduling algorithm based on task execution history. We propose a novel scheduling heuristic for the "bag-of-tasks" applications, HISA (History Injected Scheduling Algorithm). HISA uses task properties and the accumulated earlier task execution data based on a so called "similarity" factor between the individual tasks.

HISA assumes existence of a database on the applications whose runtimes and other specific properties are recorded during the previous executions. Scheduling decisions are made according to similarity between the new and earlier applications. Definition of similarity is the most important problem of this approach. On arrival of a new job, HISA defines and finds the similarity, and uses the computed similarity in assigning the job to the most suitable resource.

In order to develop, test and evaluate HISA, we need to create a Grid Scheduling Framework. Such a framework would have three parts: Resource model, application model, and scheduling policy.

In the resource model, we created nine resources with different characteristic that are available in our department and Computer Center at METU (Middle East Technical University). Each single resource was modeled after its real world counter part. Because there is no free testbed infrastructure, building such a testbed is expensive and time consuming and also in such a testbed, investigating resource management and scheduling strategies for Grid computing is harder and even near to impossible to trace. Moreover, it is impossible to create a repeatable and controlled environment for experimentation and evaluation of scheduling strategies. In a real Grid testbed, we would also encounter failures and bugs that may affect the experiments.

Secondly, we chose the application model as a BoT (Bag-of-Task). BoT applications are those applications composed of various tasks that are independent

of each other. That means at a point in time a bag of available independent tasks can be scheduled in any order as there is no inter-task communication or precedence relationships between them.

And finally, a novel scheduling heuristic HISA was developed for a bag-of-tasks applications to be scheduled on a history aware Grid. Within scope of HISA the records of task execution histories are maintained, used to find the task similarity, and then assign the job to the most suitable resource based on this similarity.

A number of experiments with various simulation settings have been conducted. Based on the experimental results, the effectiveness of our scheduling heuristic is studied and compared to the others scheduling algorithms.

The rest of this thesis is organized as follows. Chapter 2 gives detailed information about Grid computing and its properties. The Grid scheduling, taxonomy for Grid scheduling, sample static and dynamic scheduling algorithms are described in Chapter 3. Simulation tools for Grid Computing are presented in Chapter 4. In Chapter 5, history based scheduling algorithm HISA (History Injected Scheduling Algorithm) is detailed. Chapter 6 gives HISA simulation environment and results of experimentations and evaluations acquired from comparison of HISA with other scheduling algorithms. The conclusion and future work are given in Chapter 7.

CHAPTER 2

INTRODUCTION TO GRID COMPUTING

"Grid computing is a form of distributed computing that involves coordinating and sharing computing, application, data, storage, or network resources across dynamic and geographically dispersed organizations. Grid technologies promise to change the way in which organizations tackle complex computational problems in scientific, engineering, and commercial applications [44]".

Grid environment supports access to available resources in terms of seamless manner. That's why differences between platforms, network protocols, and administrative boundaries become completely transparent. Briefly, Grid is a **"**middleware" and "services" infrastructure and it is used for managing, establishing and evolving multi-organizational federation (Virtual Organizations-VO). It provides an autonomous, dynamic, and domain independent system [38].

A virtual organization can be thought as a domain. Since all resources are geographically distributed, Grid provides scalable "**virtual organizations**" in order to share these resources among such organizations. In a VO, although there is no a

central location and control, total knowledge about all systems, and trusting relationship, it persistently follows common goals. Users, information providers, and service providers like application, storage, and CPU cycle server are main elements of a VO. Actually, a VO can be thought as a network. It is not a real environment like an office. As seen from Figure 2.1, one or more resource can be shared by one or more VOs.

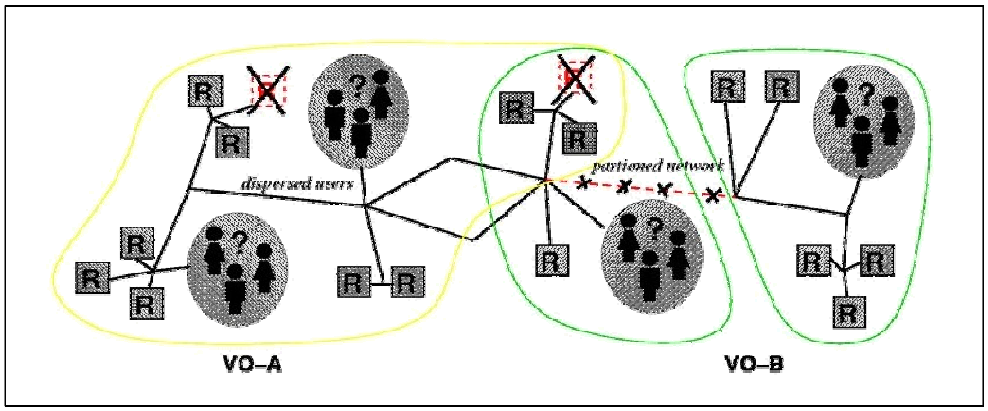

Figure 2.1: The Virtual Organization [38].

2.1 Benefit of Grid Systems

- Businesses use this technology to increase their computational power.
- Supports data sharing and distributed workflow between users
- Cooperative design and development are provided.
- Idle, available, computational and data storage resources are exploited.
- With the help of Internet, scientific research is increasingly done through distributed global collaborations between people.
- Large data collections, databases, large scale computing resources, and high performance com power can be used.

- Any grid can be easily expanded by plugging a resource to it.
- Transparency is provided to hide the complexity from the users when accessing the resources
- Making it is cheaper than the supercomputers.

2.2 Grid Architecture

Grid protocol architecture is composed of five general types of components [38]. It has layered architecture like Internet protocol architecture. Figure 2.2 shows both arcihtectures side by side. Function of each grid layer is briefly presented as follows.

1- Grid Fabric Layer
2- Grid Connectivity Layer
3- Girid Resource Layer
4- Grid Collective Layer
5- Grid Application Layer

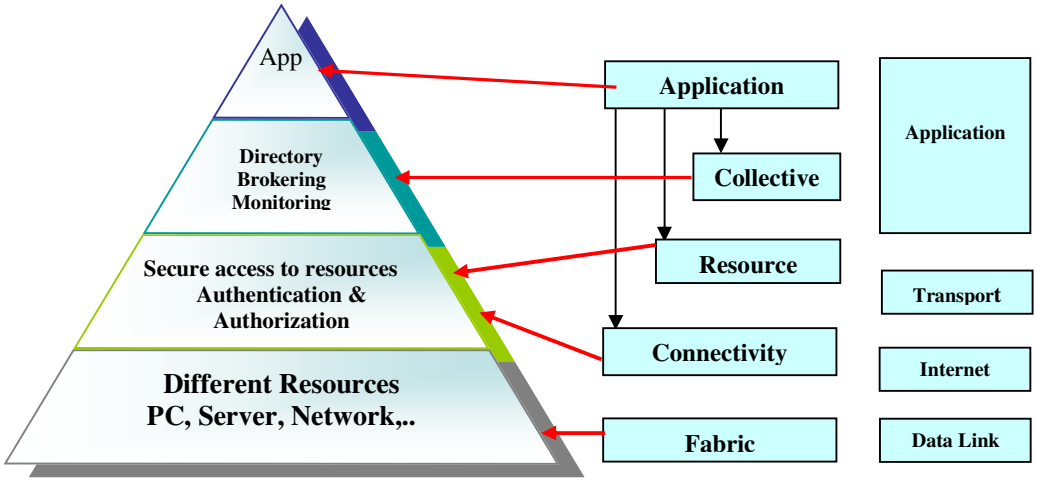

Figure 2.2: The layered Grid architecture and its relationship to the Internet protocol architecture adapted from [38].

2.2.1 Grid Fabric Layer

Fabric layer is composed of resource-specific and site-specific mechanisms. It has computers like low-end and high-end including clusters, networks, scientific instruments, and also resource management mechanisms. Examples of these mechanisms can include resource management, interfaces supporting, and advanced reservation that makes it possible for higher-level services to aggregate (co-schedule) resources in interesting ways that would otherwise be impossible to achieve some network quality of service in routers. Fabric layer also provides administer sharing operations for resources at higher levels [38].

2.2.2 Grid Connectivity Layer

It describes authentication and authorization protocols and core communication protocols for Grid-specific network transactions. These communication protocols can be used between the fabric layer resources to exchange data. Connectivity layer also provides some security mechanisims such as single sign on, delagation, user-based trust relationship [38].

2.2.3 Girid Resource Layer

The Resource layer builds on Connectivity layer and it defines the protocols (APIs and SDKs) for secure negotiation, initiation, monitoring, control, accounting, and payment of sharing operations on individual resources [38]. It also offers remote process management, co-allocation of resources, storage access, information security, and Quality of Service like resource reservation and trading [97].

2.2.4 Grid Collective Layer

Collective layer focuses on protocol and services, like API and SDKs, which are related to collections of resources. Some services which allow users to query for the availability of resources and current load are directory services. *Co-allocation, scheduling, and brokering services* allow users to request resources for their tasks scheduling on suitable resources among available resources. *Monitoring and diagnostics services* provide the monitoring of resources failure, intrusion detection, network failures, current resource load, overload, etc. *Data replication services* replicate data to increase data access performance like minimizing response time, maximizing reliability and minimizing cost.

2.2.5 Grid Application Layer

This last layer is composed of the applicaitons of users that are developed using grid-enabled languages such as HPC++, and message passing systems like MPL Specific Grid-aware application enforced with Grid Services, Grid Fabric mechanisms, and Application Toolkit components [38].

2.3 Usage Based Grid Categories

Grid systems can be grouped into seven categories according to their usage. These are as follows:

2.3.1 Traditional Grid

Traditional grid is a closed computer network that implies both its availability to only a few consumers and its being maintained by a single administrator. For example traditional grids are often build for a specific test e.g. NASA Information Power Grid [5]. It is used for official business and research that

is only open to scientists and engineers working for NASA. Traditional grids are mostly homogenous. They offer maximum performance because of their single ownership. On the other hand, these systems have minimal flexibility as they are developed for specific goals [30].

2.3.2 Modern Grid

Opposed to traditional grid, it should be open, heterogeneous and flexible. However these good properties bring some problems that need to be solved. These are secure negotiotion, authentication and authorization, sharing etc. Current Grid environments can be showed as examples for moders grid [30].

2.3.3 Computational Grid

A computational grid is consisted of resources that are specifically designed for computing power. It has mostly high-performance server. A computational Grid is a system that provides higher aggregate computational power than any single personal machine. According to how the computing power is utilized, computational Grids can be further divided into two subcategories: distributed supercomputing and high throughput. A distributed supercomputing Grid utilizes the parallel execution of applications over multiple machines simultaneously to reduce the execution time. On the other hand, the goal of the high throughput Grid is to increase the completion rate of a stream of jobs through utilizing available idle computing cycles as much as possible. A computational grid should not be thought as a parallel computing because there is no single owner responsible for all system [30].

2.3.4 Data Grid

The purpose of the data grids is to build the next generation computing infrastructure supporting intensive computation and analysis of shared large-scale databases across widely distributed scientific communities [30]. One can think data grid is a large database. However, its data is stored in different locations. In the data grid, consumers are not concerned with where this data is located and how they can access it. For example, more than two universities are working on a space research that requires a large amount of data. To overcome this problem, they can build a Data Grid to share their data, manage the data, and security issues. European Data Grid Project [36] is one example of Data Grids.

2.3.5 Scavenging Grid

Our desktop PCs form this type of grids. In scavenging grid, there is central scheduler responsible for resources. When a desktop machine becomes idle, it reports its idle time to the scheduler. For example in Seti@Home [6] project, when a desktop machine becomes idle, Seti@Home's client program enters the process and connects the machine to the grid. Then all resources of machines are working for the project. With scavenging grid, the user does not need to setup any OS or middleware like Globus. In scavenging grid, a desktop machine works for grid on a voluntary basis [30].

2.3.6 Storage Grid

A storage Grid attempts to aggregate the spare storage resources in Grid environments and provides users transparent and secure storage services [30]. For example, Network Attached Storage (NAS) and Storage Area Network (SAN) provide shared storage for multiple servers and multiple protocols. For example; UC Berkeley Storage Area Network (SAN) provides more than 30 terabytes of

Premium and Enhanced storage to more than 20 clients and expect long-term count to be closer to 100 TB. Advantages of SAN technology include increased availability, increased performance, and better monitoring through centralized administration [89].

2.3.7 Peer to Peer Grid (P2P-G)

It is based on Peer to Peer and Grid technologies and it takes advantage of both. Peer to Peer is a resource sharing mechanism available at the edge of the internet through ad hoc overlay networks with symmetric communication. A P2P computer network is a network that depends on the computing power and bandwidth of the participants in the network. In P2P, instead of creating a great network of computers, like the internet, one should be able to connect directly to the system that can provide you need. This prevents overhead. Only computers that run the same software (Chord, Pastry, BestPeer, Ares, and Emule etc.) can connect to another one. P2P can be thought as a variant of data grids because the aim of it is data exchange. Tasks are distributed to grid nodes in a decentralized manner [51]. Because of its ability and robustness, it is widely used in many cases. However, existing P2P networks do not guarantee shared data always located same place. This is not important in industries like music. On the other hand, data usage in scientific environments is different. In order to provide such a persistence and stable network P2P-G is produced [47].

2.4 Types of Grid Topologies

We can build Grids in all size whatever we want. For example, a Grid can be a few machines in a department or campus networks or group of computers organized as a hierarchy.

In general, there are three types of grid systems available according to their topologies. These are Intragrid, Extragrid and, Intergrid as seen Figure 2.3 [37].

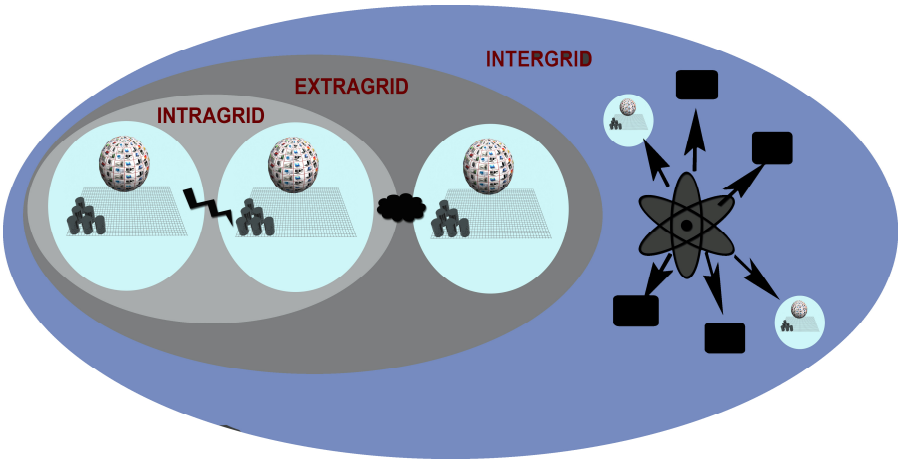

Figure 2.3: Types of Grids [50].

As seen from Figure 2.4, the simplest type of grid is Intragrid. In the Intragrid, there is a single organization with multiple departments. It provides single security management. This type of grid supports high network bandwidth. It has a simple design. The Intragrid consists of several machines that have the same operating systems and hardware component all of which are connected on a Local Area Network (LAN). It is called homogeneous systems. It is also highly secure

Intragrid has also Network Attached Storage (NAS) and Storage Area Network (SAN) to provide shared storage for multiple servers and multiple protocols. For example; UC Berkeley Storage Area Network (SAN) provides more than 30 terabytes of Premium and Enhanced storage to more than 20 clients and expect long-term count to be closer to 100 TB. Advantages of SAN technology

include increased availability, increased performance, and better monitoring through centralized administration [89].

The second type of grid is Extragrid. It is composed of two or more Intragrid. In this type of grid, there may be several partners. The Extragrid has more than one security partner.

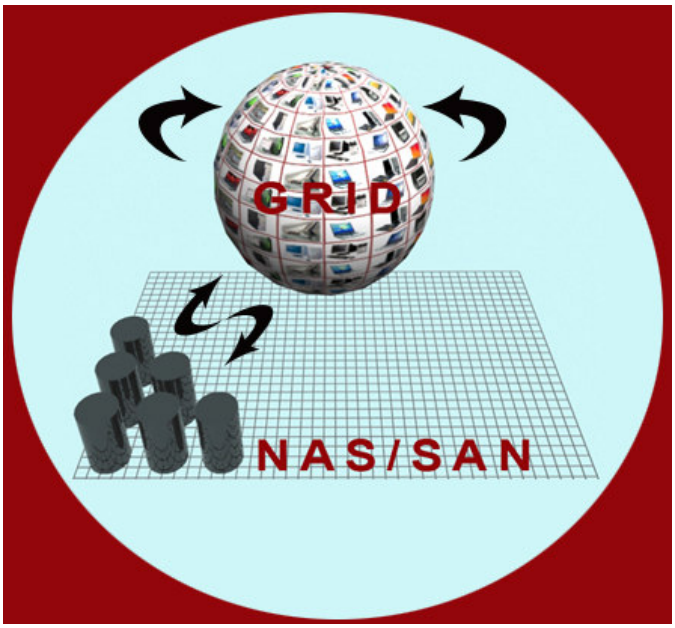

Figure 2.4: Intragrid [50].

The Extragrid consists of lots of machines that may have the same or different operating systems and hardware component all of which are connected via virtual Private Network (VPN).We can call this type of grid as heterogeneous. We can use Extragrid in a B2B (business to business) capacity or to establish the relationship of trusted business partners.

The third type of grid is Intergrid. There are many organizations and multiple partners. All Grids or Grid testbeds are put together to form the InterGrid. In an Intergrid, the data should be global data and also the applications used in this case must be modified for global users. An Intergrid can be primarily used by engineers firms, manufactures, life science industries, and by businesses in the financial industry etc.

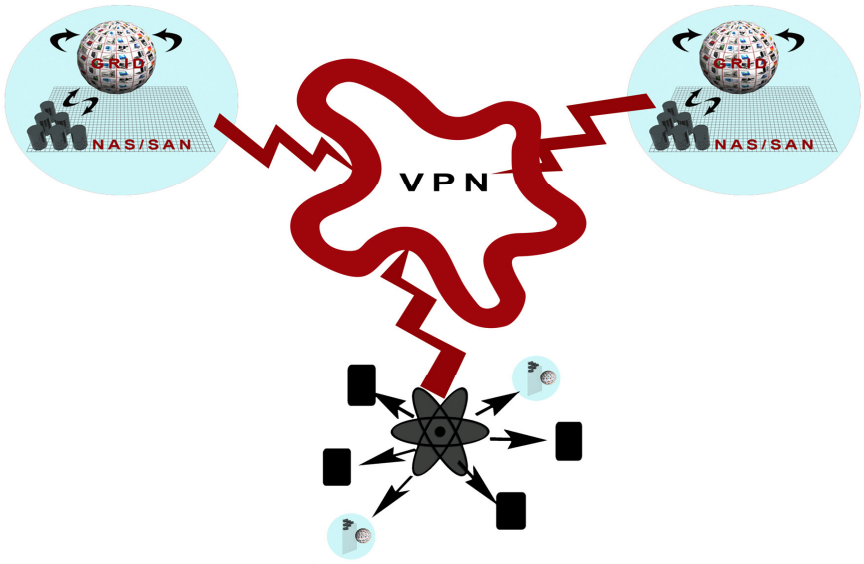

Figure 2.5: Extragrid [50]

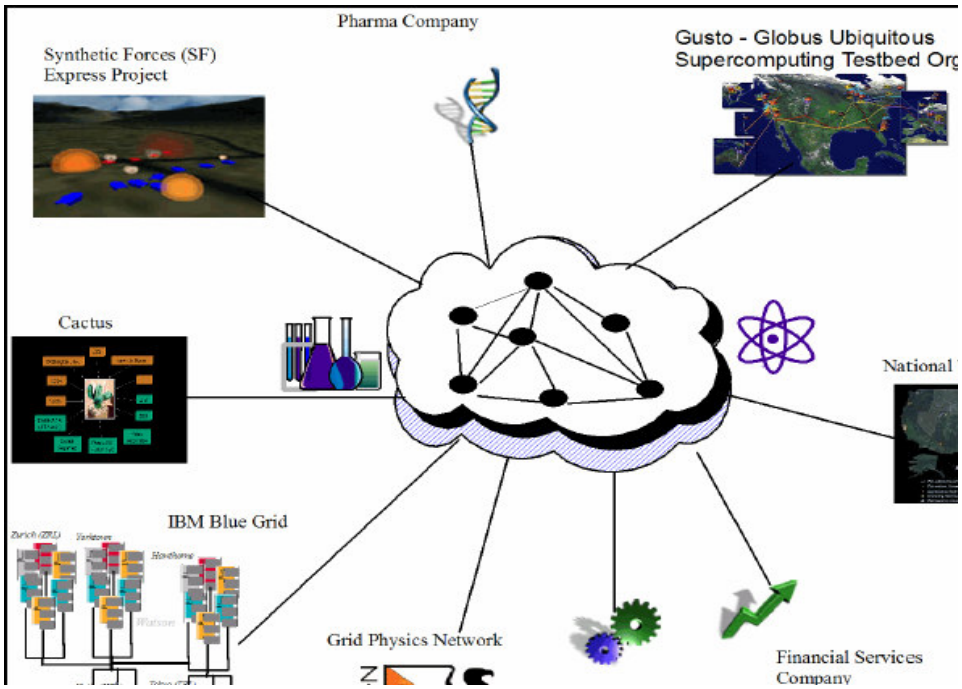

Figure 2.6: Intergrid [50]

2.5 Type of Resources in a Grid

There are several types of resources available in the grid. Computation, storage, data and databases, communications links, software and licenses, special equipment, capacities, architectures, and policies are types of resources in the grid.

2.5.1 Computational Resources (Computational Grid)

Computational resources are the most common and important resources in the grid. They can be varying in speed, architecture, software platform and connectivity. They enable CPU scavenging to better utilize resources. This means if any computer becomes idle, it reports its state to the grid. With the help of this,

users are encouraged to join to the Grid. At the end of this point, the resource used for Grid. Computational grid aggregates the processing power from a distributed collection of systems. They provide the computational power to process large scale jobs. Computational resources satisfy the business requirement for instant access to resources on demand [37].

In the grid system there are three major ways to utilize the computation resources:

1. An existing parallel application can be run on the grid.
2. Applications or tasks can be divided into separate parts and execute in parallel on different machines in the grid.
3. Application is run many times on many different machines at the same time.

2.5.2 Data Storage Resource (Data Grid)

It is the second most common resource in the grid. It can be memory attached to the processor or it can be secondary storage such as HDD and type driver to increase capacity, performance, sharing and reliability of data. Data grid provides a scalable storage and access to the datasets. Replicated, catalogued, and even different datasets are stored in different locations to create an illusion of mass storage. Using unified file system such as Andrew File System (AFS) and Network File System (NFS) with the storage on multiple machines increase the capacity. These advanced file systems can duplicate sets of data. Meanwhile, an intelligent grid scheduler can help chosen suitable storage device to hold data depending on usage job patterns. Moreover in the grid system, journaling can be implemented by the grid file system so that the data can be recovered in a more reliable way after the failures. Since data is shared and updated by lots of users, grid file system performs advanced synchronization mechanism to reduce contention between these users.

Also data striping in writing or reading successive records to/from different physical devices overlap the access for faster throughput [11].

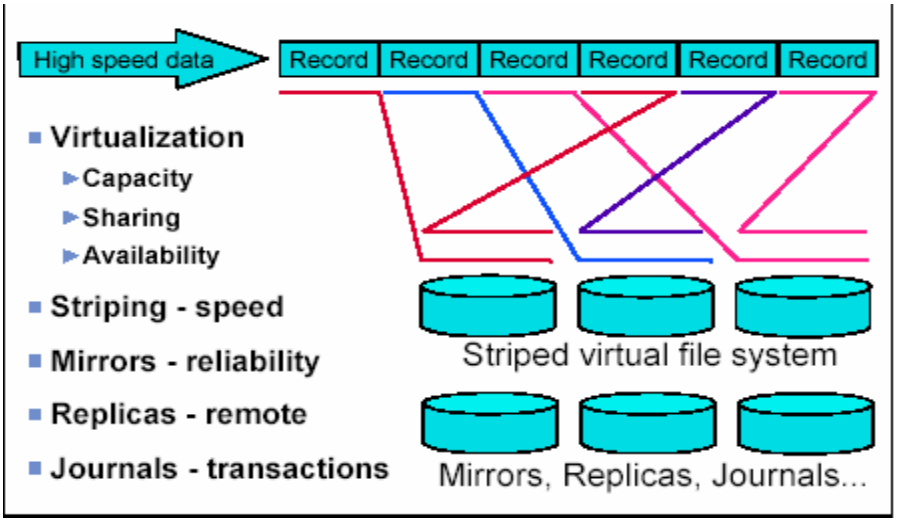

Figure 2.7: Data Grid [11].

2.5.3 Communication

It becomes another resource in the grid when some jobs require a lot of data to be processed since bandwidth can be critical resource that can limit utilization of the grid for such jobs. Sometimes to overcome potential network failures and huge data traffic, redundant communication paths e.g. VPNs are needed. In an Intergrid, if it is assumed that we want to develop a search engine that should access the external Internet to provide connectivity among the grid machines, in this case, these connections do not want to share the same communications path, and then they want to add the new total available bandwidth for accessing the Internet [11].

Software and licenses, special equipment, capacities, architecture and policies represent a different kind of resources. Installation of a too expensive software on every grid machine increases cost. To prevent this, this software is installed on some particular machines that jobs requiring this software to be sent. Therefore this method can decrease the cost for an organization. On the other hand, some software licensing arrangements allow the software to be installed on all of the machines but may limit the number of software instances that can be simultaneously executed at any given time. This limitation is enforced by license management software [37].

Because of the heterogeneous and the dynamic nature of the grid, it has often different architectures, operating systems, devices, capacities, and policies. Each of these items represents a different kind of resource while the grid assigns jobs to machines since it can use this **special equipment, capacities, architecture and policies** as criteria. For example there is several type of software running on different architecture such as x86, SGI origin, Sun Ultra etc. Therefore users must consider such characteristics when assigning the jobs to machines in the grid [37].

In general, we can classify Grid resources as shown in Figure 2.8.

A Grid resource can have resource ID, resource name, cost (price), and performance criteria.

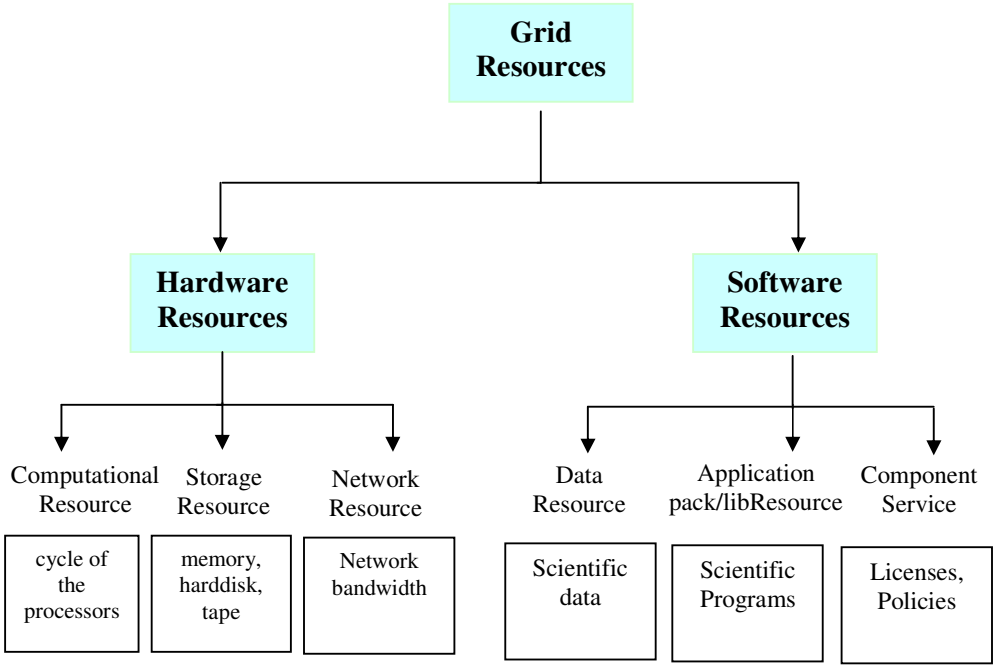

Figure 2.8: A classification of Grid resources

CHAPTER 3

RELATED WORK

3.1 Grid Scheduling

The management of grid resources in a grid environment is clearly important. The overall aim is the efficient and effective scheduling of the applications that need to utilize the available resources in the distributed environment. Grid scheduling is essential to provide consistent and coordinated use of heterogeneous Grid. From a user's point of view, resource management and scheduling should be transparent.

Because of the heterogeneous and the dynamic nature of the grid, scheduling is significantly complicated due to the difference in performance goals by various grid applications (users) and grid resources. Most grid systems use grid scheduler responsible for resource discovery and available resources, by selecting the most suitable resources that provide user's request, and assigning the task onto selected resources.

A grid system responsible for executing a job does this in two ways: *In the simplest of grid systems,* the user selects any machine or any resource suitable for running his job. On the other hand, in a *more advanced grid systems,* the job scheduler finds the most appropriate machine to run a given job [11].

There have been lots of scheduling algorithms introduced for homogeneous and dedicated resources such as computer clusters. Since clusters have monotonic performance goal, high speed interconnection network, dedicated resources, homogeneity of resources and applications, these scheduling algorithms (e.g. First Come First Serve, Minimal-Requested-Job-First, Shortest Job First, Backfill and so on) have a complete control over all computing nodes and the overall information about the pending jobs. Thus, the scheduler can schedule jobs onto the cluster effectively and efficiently. However these algorithms cannot be suitable and work well for today's heterogeneous environments such as Grid. Therefore, Grid must deal with large-scale heterogeneous resources across different management boundaries. In such a dynamic distributed computing environment, resources availability varies dramatically. For example, while some resources may go offline because of some network and hardware failure, others may go online. Scheduling in Grid environments is significantly more complicated than the traditional scheduling systems for distributed environments. Grid scheduling systems must take diverse characteristics of both various Grid applications and various Grid resources into account. Therefore, scheduling becomes quite challenging in Grid [59].

Ordering and mapping are two deterministic characters in grid scheduling. Ordering is applied when there are more than one tasks waiting for execution to determine by which order the pending applications are arranged. Ordering must be done according to applications' priority or deadline factors. That means access to resources is typically subject to individual access, priority, and deadline policies of the resource owners. On the other hand, mapping is finding the most suitable resources and then assigning the applications to such resources. For the best

scheduling, the performance potential should be estimated for each mapping [98].

Scheduling in grid computing means that one or more users' tasks that can be submitted without knowing where the resources are or even who own the resources. A Grid scheduler or scheduling algorithm has to guarantee the quality of service of a job's execution. In this context the following terminology is used [35]:

- A **task** is a single indivisible unit to be scheduled by any grid schedulers.
- The **properties** of a task are parameters like length, deadline, priority, CPU requirement etc.
- An **application** (or **job**) is composed of more than two tasks and each task has sub-tasks that have an ability to decompose themselves further into atomic tasks
- A **resource** is something that is required to carry out an operation, for example; computational resource, network resource, software resource, data, database, instrument, code repository and storage space.
- A **node** (or **site**) is composed of one or more grid resources.

Each job consists of a number of tasks. While scheduling this task to resources, we encounter several challenges:

- Resource heterogeneity: How does the heterogeneity of resources affect the performance of a schedule?
- Site Autonomy: How do the site's scheduling policy, local priority, and security and management policy affect the quality of a schedule?
- Non-Dedicated Resources: Here contention is a major issue and it can exist both on computational resources and network connections. Moreover a resource may join multiple Grids simultaneously if it is non-dedicated.
- Application Diversity: In Grid, applications are from wide range of users

each of whom has its own special requirements.

- Dynamic Behavior: In Grid, some resources may go off-line and some others may come on-line, any time.

Aim of the scheduling algorithms is to minimize make-span, idle time of the available resources, turn-around time and meet the specified application deadlines.

According to Casavant et al [23], the scheduling problem consists of three main components: consumers, resources and policies. 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. The policy chosen for implementing a scheduler affects both users and resource providers.

A scheduler is designed to satisfy one or more of the following common objectives:

- Maximizing resource utilizations,
- Maximizing system throughput (it is the amount of work that a computer can do in a given time period),
- Maximizing economic gains (for system/resource owner perspective),
- Minimizing the turn-around time for an application/job/metatask,
- Minimizing the overall execution time of a job,
- Balance the load among the grid resources,
- Minimizing the makespan is the total time that lasts until the last task in the job is done,
- Minimizing the processors cost and economic cost (from the users/consumers perspective),
- Minimizing the communication cost,
- Meet the users' QoS requirements (from the user perspective),

- Meet the users' deadline and budget request (from the user perspective).

It is also possible to classify the scheduling according to their primary object [2].

1- Application Centric

a- Makespan
b- Economic Cost

2- Resource Centric

a- Resource Utilization
b- Economic Profit

In the most general case, scheduling algorithms have to be adapted to the different optimization criteria that a user can specify for each particular job. Some of the most frequent optimization criteria for a user are the following [64]:

1- Optimizing performance without regard to the cost.
2- Optimizing cost without regard to the performance
3- Optimizing performance within a specific time constraint.
4- Optimizing cost within a specific time constraint.
5- Optimizing performance within a specific time and cost constraint.
6- Optimizing cost within a specific time and cost constraint.

3.2 Taxonomy for Grid Scheduling Algorithms

In general, Casavant et al [23] introduce a hierarchical taxonomy for scheduling algorithms and distributed computing systems. Since Grid scheduling is

similar to this system, it can be thought a subset of this taxonomy. Our taxonomy is based on this approach as shown in Figure 3.1.

In local scheduling also called centralized or cluster scheduling, there is a central leader (main scheduler) responsible for scheduling. This type of scheduling can only support uniform policy and suits well for cluster management systems such as Condor [31], LSF (Load Sharing Facility) [41], Beowolf [72] and Codine [83] etc. While the centralized scheduling algorithms are implemented easily, they may not support efficient load balancing, fault tolerance, and scalability. The centralized scheduling algorithms are not suitable for grid resource management systems because grid systems are expected to abide by the policies that are imposed by resource owners. However, the centralized scheduling algorithms do not allow local policies applied by local resource owners. Therefore local resource owners do not want to join such organizations.

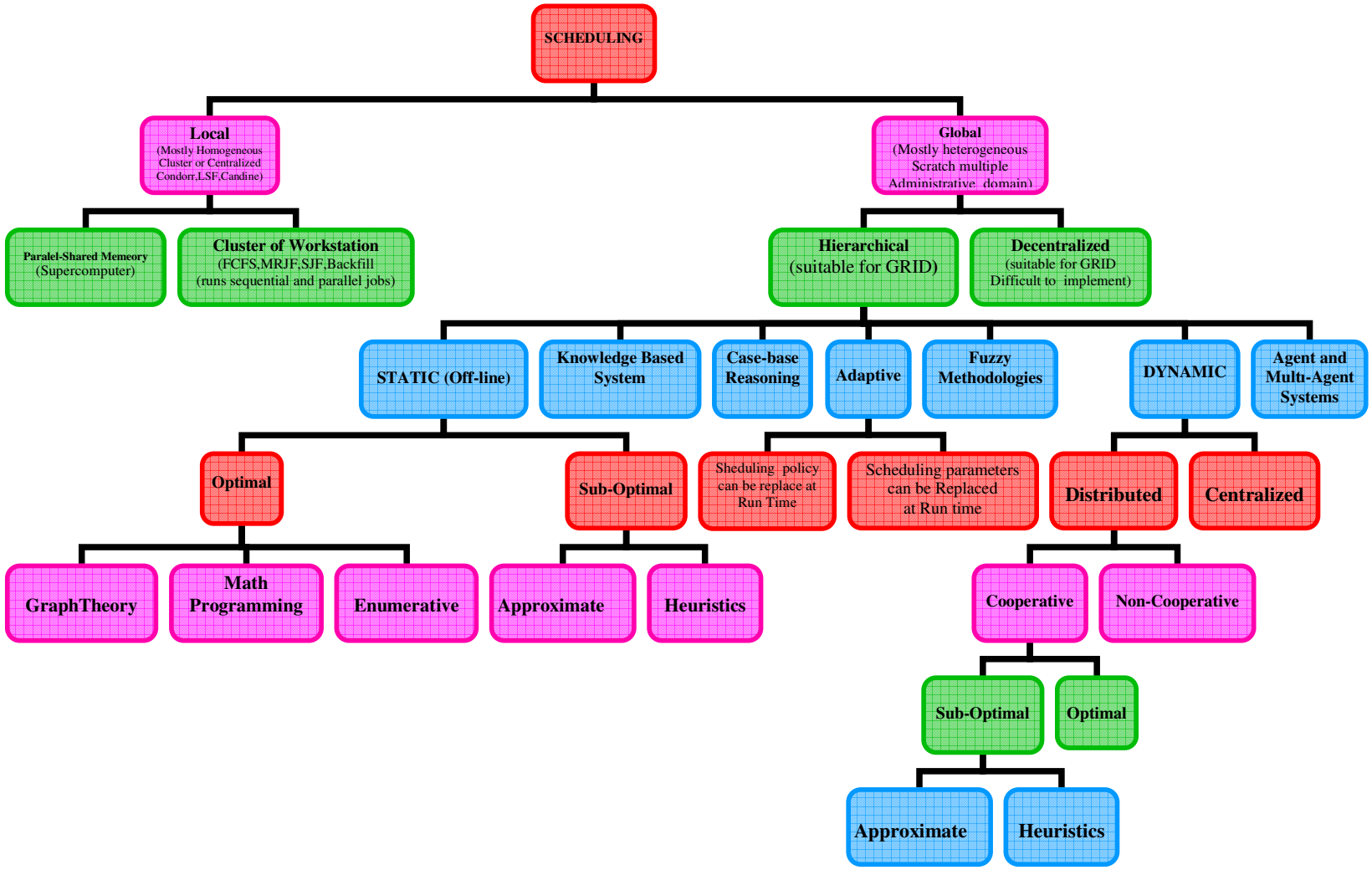

Figure 3.1: A Taxonomy for Grid Scheduling adapted from [23]

We can divide global scheduling into two sub categories. One of them is decentralized scheduling model. In this model, there is no central leader responsible for scheduling. Therefore we do not encounter disadvantages of centralized scheduling model. Decentralized scheduling strategy is suitable for grid systems because resource owners can define their scheduling policies that schedulers can enforce. However as the resource owners may not agree on a common policy goal for resource management, development of scheduling algorithms seems to be difficult.

The hierarchical scheduling technique is the most suitable scheduling scheme for grid systems because this model permits the remote resource owners to enforce their own local policy on external users. That means different schedulers may take independent decisions at each level of hierarchy [97].

Hierarchical scheduling can be divided into seven sub-categories. These are knowledge-based system, case-base reasoning, fuzzy methodologies, adaptive, agent and multi-agent systems, static, and dynamic scheduling.

The aim of the knowledge-based system is to capture the experience of the scheduling expert. In the knowledge-based system, there is an interface used to gain conclusions or recommendations regarding the scheduling problems [84].

Case-Based Reasoning (CBR) scheduling is an artificial intelligence method. In the CBR scheduling, reusing knowledge and experience earned in solving previous problems are used to solve a new problem. A description of the problem and its solution are saved as a case. All cases are put in a case base. The CBR process has four phases: take the case most similar to the new problem, reuse and revision of its solution, and inclusion of the new case in the case base [55].

Fuzzy methodology is another traditional scheduling technique. Scheduling models that use fuzzy methods have recently attracted more attentions on scheduling research communities [79]. In a busy Grid environment, since scheduling needs to be a continual, dynamic and uncertain process, SLAs (Service-Level Agreements) are constantly added, altered or withdrawn. Some preliminary work has been carried out to examine whether fuzzy methods can be used in the evaluation of Grid performance contract violations, however this has been very simplistic because it is based on 2 fuzzy rules with 2 variables [90].

Adaptive scheduling technique is suitable for dynamic structure of Grid because it dynamically evaluates application efficiency and its execution times, and the number of resources is adapted with the help of this information and finally the applications are assigned to the resources [46]. According to the author in [43], with adaptive scheduling technique scheduling strategy and parameters can be replaced at runtime.

The researches have showed that to overcome a wide range of complex distributed scheduling problems, Multi-agent systems may have been used successfully. Since Multi-agent systems both have autonomous and distributed and dynamic nature, they are robust against faults. Therefore, we can build complex, robust, and cost-effective next-generation scheduling systems with Multi-agent systems. In [20], authors claim that the foundation for the creation of Grid scheduling systems that have capabilities of autonomy, heterogeneity, reliability, maintainability, flexibility, and robustness can be provided by the Multi-agents systems.

Briefly, it seems hierarchical scheduling is the most useful technique for the development of scheduling algorithms. We can classify hierarchical scheduling algorithms according to the system's being dynamic or static, the dynamic case can further be classified as distributed and centralized, as shown in Figure 3.1.

In this thesis, we will focus on both dynamic and static scheduling techniques.

3.3 Static Scheduling

In the static scheduling also called off-line scheduling technique, all decisions are done before the execution of a schedule. It is suitable when all the tasks (or applications) and resources are known in advance. Every task is assigned once to a resource. Because both the foresight of knowledge of the task to be assigned and the cost estimation of the tasks being simple facilitates, the development of algorithm in static scheduling is so simple. However, this technique which uses cost estimation based on static information is not appropriate to situations where dynamic changes are significant and frequent. [35].

Static scheduling algorithms can be examined in two sub groups: Optimal and sub-optimal. However, because of the NP-COMPLETE nature of such scheduling algorithms and complication of the implementation of the grid scenarios, optimal scheduling sounds unpractical. Hence, current research concentrates on sub-optimal scheduling algorithms to some extent.

Sub-optimal static scheduling algorithms can be divided into two subgroups:

- Approximate and
- Heuristics

An algorithm returning a near-optimal solution is called approximate algorithm. This technique does not guarantee the best solution. An approximate algorithm is satisfied when a solution that is sufficiently “good” is found. The goal of an approximation algorithm is to come as close as possible to the optimum

schedule in a reasonable amount of time. One example of such algorithm is TPCC (Total Processor Cycle Consumption*)* [42].

While approximate algorithms give near-optimal schedule, heuristics algorithms find solutions among all possible ones, but they do not guarantee that the best will be found. Therefore these algorithms can be used to find a solution close to the best one and they find it fast and easily. Due to this feature of heuristic algorithms and heterogeneity and dynamic structure of grid, they are more practical for grid scenarios.

3.3.1 Examples of Static Algorithms

Static scheduling heuristics are based on static processor assignment disciplines. It is priority list scheduling that ranks the tasks in a priority list according to a given heuristic. The task with the highest priority is assigned to a processor according to a heuristic. In this approach, a task graph is used to represent the tasks and their properties such as level and service demand of tasks which is the average time of the task to execute on the fastest processor as shown in Figure 3.2 [62].

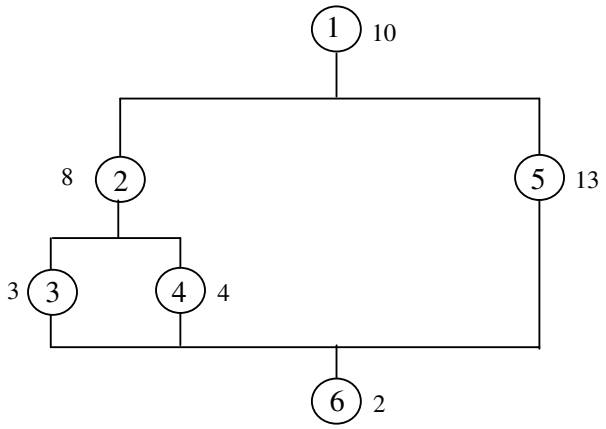

Figure 3.2: A task graph

Level of task is computed as follows.

$$l(t) = \max_{\pi \in \Pi(t)} \sum_{t_k \in \pi} sd(t_k)$$

where:

sd is service demand of a task. It is the average time of the task to execute on the fastest processor.

Π (t) is the set of paths in the task graph which start from task t and to the last task.

l(t) is level of a task. It is defined as the maximum of the sum of the sd.

For example, let service demand of tasks 1 through 6 be 10,8,3,4, 13, and 2 respectively as shown in Figure 3.2. According to this information and using formula 1, we can compute level of task 1 as follow:

$l(1)$ = max [path1(1,5,6), path2(1,2,4,6), path3(1,2,3,6)]

This is over all the paths from task 1 to task 6.

$l(1)$ = max [(10+13+2),(10+8+4+2),(10+8+3+2)]

= 25

Weighted level of a task is computed as follows.

$$wl(t) = sd(t) + \max_{t_{ii} \in succ_G(t)} wl(t_i) + \frac{\sum t_i \in succ_G(t)\ wl(t_i)}{\max_{t_{ii} \in succ_G(t)} wl(t_i)}$$

where:

$succ_g$ (t) is the set of immediate successors of t in the task graph.

wl(t) is weighted level of a task. It is the sum of the service demand of a task with the maximum of the weighted levels of all its successors plus the sum of the weighted levels of the successors of t normalized by the maximum weighted level among all successors of t.

In Figure 3.2, the weighted level of tasks 1, 2, and 5 are 85.85, 16.86, and 16 respectively.

In [62], Menasce et al proposed a static heuristic processor assignment algorithm. Their algorithm is composed of two parts:

- Envelope,
- Heuristic.

First the subset of tasks called task domain and the subset of processor called processor domain are selected by the Envelope at each step. And then an appropriate (task and processor) pair is chosen by the heuristic from the task and processor domain respectively. Therefore, a combination of Envelope and Heuristic represents a static heuristic processor assignment algorithm.

Three static task scheduling heuristics can be given as examples:

HLF (Highest Level First) chooses the task with the highest level (*max l(t)*)

LTF (Largest Task First) selects the task with the largest value of service demand

WLF (Highest Weighted Level First) chooses the task with the highest value of weighted level.

Whenever a task is selected from any one of the above heuristic, the most suitable processor is selected according to any of the following processors disciplines:

SEET (Shortest Estimated Execution Time) selects the processor to execute the selected task fastest.

MFT (Minimum Finish Time) chooses the processor from the processors domain that minimizes the completion time of selected task.

So, static scheduling heuristic are combination of task and processor selection disciplines.

From this perspective, some **static grid scheduling** algorithms are as follows.

3.3.1.1 MET (Minimum Execution Time)

It assigns each task, in arbitrary order, to the next machine with the best execution time for that task regardless of the availability of that machine. The goal of the MET is to give each task its best machine. However it does not consider the availability time of the machine. Therefore, this causes a severe load imbalance across machines. Algorithms complexity is O(m) where m is the number of machines in heterogeneous environment. This algorithm is not suitable for grid environment where resources and tasks are characterized as consistent, that means if a resource runs a task faster, it will run all the other tasks faster [39].

3.3.1.2 OLB (Opportunistic Load Balancing)

It is also known First Come First Serve (FCFS). It is commonly used for task scheduling. In contrast to the MET, OLB assigns each task in arbitrary order to the next available machine regardless of the expected execution time of the task [12]. However OLB keeps all machines quite busy. While implementation of OLB is simple, it gives poor makespan which is the total time that until the last task in the metatask is done because it does not consider expected task execution time [7], [40].

3.3.1.3 MCT (Minimum Completion Time)

It assigns each task, in arbitrary order, to the machine with the minimum expected completion time for that task.

It is only designed for independent tasks assignment and suitable for space-shared systems. The aim of the MCT is to minimize the makespan of a set of independent tasks. However, while MCT does this, it causes severe load on the systems [19].

$$ct = bt + et$$

where;
ct is completion time of a task
bt is beginning time of a task on the machine
et is the execution time of a task

The MCT joins the benefits of MET and OLB. This combination prevents circumstances in which MET and OLB perform poorly. Complexity of the algorithm is also O(m), where m is the number of machine in grid. MCT is currently used in NetSolve [8].

3.3.1.4 MAX-MIN

In Max-Min heuristic, U is the set of all unassigned tasks. Firstly, the set of minimum completion times, M = $\{\min_{0\leq j<u} (ct (t_i, m_j))$, for each t_i € U$\}$, is found. Next, the task with the overall maximum completion time from M is selected and assigned to the corresponding machine (Hence the name Max-Min). Finally, the newly mapped task is removed from U, and process is repeated until U is empty, that is, all tasks are assigned [48].

The advantage of the Max-min is to minimize the problems incurred from performing tasks that have longer execution times. Assigning the task to the best machine which has the longer execution time allows this task to be executed concurrently with the remaining tasks which have the shorter execution times [12]. The dynamic version is also available. This will be explained in dynamic scheduling heuristic section.

for *all task* t_i *in meta-task* M_v *(in an arbitrary order)*
    for *all machines* m_j *(in a fixed arbitrary order)*
        $ct_{ij} = et_{ij} + r_j$
do *until all task in* M_v *are mapped*
    for *each task* t_j *in* M_v *find its earliest completion time and the machine that obtains it*
    find *the task* t_j *with the* ***maximum*** *earliest completion time*
    assign *the task* t_i *to the machine* m_i *that gives the earliest completion time*
    delete *the task* t_k *from* M_v
    update r_l
    update ct_{il} *for all* t_i € M_v
enddo

Figure 3.3: The Max-Min Heuristic

Where;

ct_{ij} is the *completion time of task* t_i on machine m_j

et_{ij} is *the expected execution time of task* t_i on machine m_j

r_j is next available or *ready time* of machine m_j

3.3.1.5 MIN-MIN

It is similar to the Max-Min. It calculates the minimum completion time for each task in the application that is currently being mapped. For example, U is the set of all unassigned tasks. First the set of minimum completion times, M = $\{\min_{0\leq j<u} (ct(t_i,m_j))$,for each $t_i \in U\}$, is found. Next, the task with the overall minimum completion time from M is selected and assigned to the corresponding machine (Hence the name Min-Min). Finally, the newly mapped task is removed from U, and procedure is repeated until U is empty, that is, all tasks are assigned [48]. Min-min is developed on MCT, but while MCT takes only one task into consideration at a time, Min-min takes all unassigned tasks into consideration during each mapping decision.

Min-Min firstly maps the task t_i on an empty system. Assume that m_j is a machine that finishes t_i earliest and also executes it fastest. After assignment of t_i, for every task, the heuristic changes the availability status of m_j by the least possible amount for every assignment. Therefore the percentage of tasks assigned to their first choice (on the basis of execution time) is likely to be higher than Max-min. That means if more tasks are mapped to the machines that complete them the earliest and also execute the fastest; a smaller makespan can be obtained. However, since Min-min first finishes the shorter tasks and then executes the longer one, it increases the makespan compared to the Max-Min. The dynamic version is also available. This will be explained in dynamic scheduling heuristic section.

for *all task* t_i *in meta-task* M_v *(in an arbitrary order)*

  for *all machines* m_j *(in a fixed arbitrary order)*

    $ct_{ij} = et_{ij} + r_j$

do *until all task in* M_v *are mapped*

  for *each task* t_i *in* M_v *find its earliest completion time and the machine that obtains it*

  find *the task* t_i *with the* ***minimum*** *earliest completion time*

  assign *the task* t_i *to the machine* m_i *that gives the earliest completion time*

  delete *the task* t_i *from* M_v

  update r_l

update ct_{il} *for all* t_i M_v

enddo

Figure 3.4: The Min-Min Heuristic

Both Min-Min and Max-Min are implemented in SmartNet [40].

3.3.1.6 QoS GUIDED MIN-MIN

In Grid computing, QoS may show differences in terms of different types of resources. Sometimes it can be CPU speed (MIPS), network bandwidth, or sometimes utilization of CPU, or storage capacity.

QoS Guided Min-Min is a version of the Min-Min. It both quarantines the QoS requirements of particular tasks and minimizes the makespan. In the Grid systems, a task can be sent to a resource with or without QoS requirements. For example, if we send a task without QoS request, it may be assigned on both high and low QoS resources. On the other hand, if we send a task with high QoS request it can only be executed on a high QoS resource. Therefore, if a task with low QoS is assigned to a resource with high QoS, other task with high QoS will wait until the

resource with high QoS remain idle. To prevent this shortcoming, in [93], authors modify the Min-Min heuristic and develop a QoS Guided Min-Min scheduling heuristic.

In this QoS guided Min-Min heuristic, authors consider the matching of the QoS request and service between the tasks and hosts based on the conventional Min-Min. The algorithm consists of the initialization of the completion time, and two "do" loops that schedule the high QoS tasks and low QoS tasks, respectively as seen Figure 3.5.

for *all tasks* t_i *in meta-task* M_v *(in an arbitrary order)*
    for *all hosts* m_j *(in a fixed arbitrary order)*
        $ct_{ij} = et_{ij} + r_j$
do *until all tasks with high QoS request in* M_v *are mapped*
    for *each task with high QoS in* M_v*, find a host in the QoS qualified host set that obtains the earliest completion time*
    find *the task* t_k *with the minimum earliest completion time*
    assign *task* t_k *to the host* m_l *that gives it the earliest completion time*
    delete *task* t_k *from* M_v
    update r_l
    update ct_{il} *for all i*
end do
do *until all tasks with low QoS request in* M_v *are mapped*
    for each task in M_v *find the earliest completion time and the corresponding host*
    find the task t_k *with the minimum earliest completion time*
    assign task t_k *to the host* m_l *that gives it the earliest completion time*
    delete task t_k *from* M_v
    update r_l
    update ct_{il} *for all i*
end do

Figure 3.5: The QoS Guided Min-Min Heuristic

3.3.1.7 SEGMENTED MIN-MIN

In the Segmented-Min-min (S-Min-min) algorithm, tasks are sorted according to ETC (Expected Completion Time) Matrix. The tasks can be sorted into an ordered list by the average ETC, the minimum ETC, or the maximum ETC. Then, the task list is partitioned into segments with the equal size. The segment of larger tasks is scheduled first and the segment of smaller tasks last. For each segment, Min-min is applied in order to assign tasks to machines. The difference between the Min-min and Segmented-Min-min algorithms is that Segmented-Min-min performs task sorting before scheduling. The goal of the sorting means that larger tasks are promoted to be scheduled earlier. Then, Min-min is applied locally within each segment. The advantage of the Segmented-Min-Min is if the lengths of the tasks are strikingly different, the segment improves the performance [92].

3.3.1.8 SUFFERAGE

Before we explain the heuristic, we should define the sufferage value. Sufferage value of a task is defined as the difference between its best minimum completion time (MCT) or earliest completion time on a machine m_A and second best MCT on a machine m_B. Hence we can say the sufferage heuristic maps each task based on the MCT. In the sufferage heuristic, a task can be assigned to a machine according to its sufferage value. The heuristic assigns the task with the highest sufferage value first. That means the task with the high sufferage values takes precedence [60].

However when there is input and output data for the tasks and resources that are clustered, this sufferage heuristic may have problem. For example a task t_i with large input file is already stored on a remote cluster. If this cluster contains two (or more) hosts with almost alike performance, then each of this can achieve nearly the same MCT for that task. Therefore these two hosts lead to give almost similar the

best and second-best MCTs for t_i. This means that the sufferage value will be close to zero and this gives the task low priority. On the other hand, other tasks may be mapped in its place. Since this generates load on the hosts in the cluster, t_i is forced to be scheduled on some other clusters. Therefore this requires an additional file transfer and eventually causes large makespan because of poor file reuse [21].

```
for all task t_i in meta-task M_v (in an arbitrary order)
    for all machines m_j (in a fixed arbitrary order)
                ctij = etij + rj
do until all task in M_v are mapped
        mark all machines as unassigned
        for each task t_k in M_v (in an arbitrary order)
                find machine m_j that gives the earliest completion time
                sufferage value = best completion time – second best completion
                time
                if machine m_j is unassigned
                        assign t_k to machine m_j delete t_k from M_v, mark m_j
                        assigned
                else
                    if sufferage value of task t_i already assigned to m_j is less than
                    the sufferage value of task t_k
                        unassign t_i,, add t_i back to M_v,
                        assign t_k to machine m_j,
                        delete t_k from M_v
        endfor
        update the vector r for the tasks that were assigned to the machines
        update the c matrix
enddo
```

Figure 3.6: The Sufferage Heuristic

3.3.1.9 XSUFFERAGE

It is introduced by Casanova et al [21]. The purpose of XSUFFERAGE is scheduling applications by taking data transfers into account. Casanova showed that the Sufferage heuristic is not suitable for cluster type of resources. The MCT on the machines belonging to the same cluster might be quite close to each other. This situation closes the sufferage value to zero and cluster machines are eliminated from the selection [21]. The XSUFFERAGE uses cluster-level MCT, that is used to calculate the sufferage value, rather than the MCT in sufferage heuristic. That is, MCTs are computed for tasks on each cluster then sufferage values are computed for each cluster and finally SUFFERAGE heuristic is applied to each of them. Casanova et al showed that XSUFFERAGE defeats the SUFFERAGE not only in the large data files, but also when the resource information cannot be predicted very accurately. Contrary to this, XSUFFERAGE uses lots of dynamic information about the grid environment and the applications. Since it is hard to acquire this type of information, this heuristic sometimes gives poor solution.

3.3.1.10 TASK GROUPING

It is introduced by Muthuvela et al [66]. The aim of the Task Grouping heuristic is to cope with metatasks which do not have connection with each other and cause a large number of lightweight jobs requiring a high overhead cost when scheduling and transmitting the applications to or from Grid resources. Whenever a set of fine-grained tasks (“Fine-grain”, or “tightly coupled, parallelism” means individual tasks are relatively small in terms of code size and execution time. The data are transferred among processors infrequently) are received, the scheduler groups them into a few job groups according to their requirements for computation and processing capability of available Grid resources that can provide this in a certain time period. After that all tasks in the same group are assigned to the same resource which can finish them in the given time period. Therefore while task

grouping heuristic increases the resources utilization, it also decreases the overhead and job launching time (makespan) [66].

3.3.1.11 SA (Switching Algorithm)

It is created from the combination of MET and MCT. SA uses the MCT and MET in a switching fashion according to the load distribution across the machine. For example; The MET heuristic may cause load imbalance between the resources by mapping many more tasks to the same machine than the others. On the other hand, the MCT heuristic attempts to prevent load imbalance by mapping tasks according to the earliest completion time. For example r_{max} is the maximum ready time over all machines in the grid environment and vice versa r_{min} is the minimum ready time. Then, the load balance index across the machines is given by $\pi = r_{min}/r_{max}$. The parameter π can be interval [0, 1]. Two threshold values, π_l (low) and π_h (high), for the ratio π are chosen in [0, 1] where $\pi_l < \pi_h$. First π is set to 0.0. The heuristic begins with the MCT that assigns tasks according to minimum completion time until the value of load balance index increases to at least π_h. As soon as reaching this point, the SA heuristic starts the MET heuristic to map the tasks on suitable resources. This decreases the load balance index until it reaches π_l, this cycle continues until all tasks are mapped [76].

3.3.1.12 GA (Genetic Algorithms)

It is one of the Nature's Heuristics. GA is based on genetic process of biological organisms. It is an evolutionary technique used for large solution space to find near-optimal solution. If we suitably code the GA, we can use them in solving the real world problems.

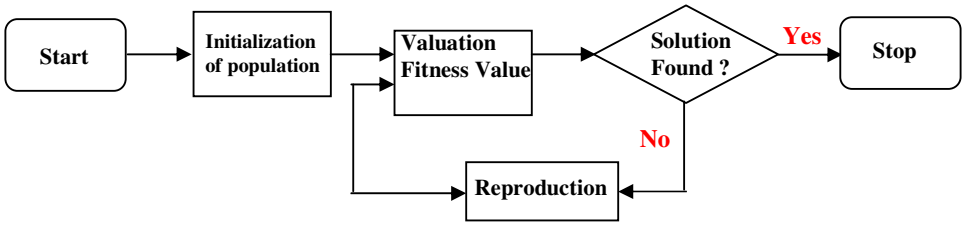

Figure 3.7: Iteration of GA [1]

The GA consists of four important phases.

The first phase of the GA algorithms is population generation. A population consists of a set of chromosomes. Each chromosome defines a possible solution that assigns a task to a resource. Firstly, all chromosomes are randomly generated from a uniform distribution. Alternatively one chromosome is generated using by other scheduling heuristic like Min-Min, and the other chromosomes are generated randomly. This method is also called seeding the population with a Min-min chromosome.

The second phase is chromosome evaluation. Each chromosome has a fitness value. This fitness value represents the makespan which is the total time that until the last task in the metatask is done (matching and scheduling). Therefore we can say the goal of the GAs is to acquire the smallest makespan.

The third phase is crossover and mutation operation. In the crossover operation, a random pair of chromosomes is chosen and then a random point is selected in the first chromosome. For the sections of both chromosomes from that point to the end of each chromosome, machine assignments are exchanged by the crossover between corresponding tasks. After the crossover operation, a chromosome is randomly selected by the mutation operation. And then a task within the chromosome is randomly selected and assigned to the new machine again

randomly. Every chromosome is considered for mutation with some probability. A chromosome is transformed by the mutation process another valid one that may or may not already be in the current population. And finally, one iteration of the GA is completed with the chromosomes from this modified population is evaluated again. When a predefined number of evolutions are reached or all chromosomes cause the same mapping, the GA is stopped [91].

Several types of GA algorithms are developed for scheduling algorithms in literature. Some of them are found in [96], [3], and [52].

In summary, the steps that are taken to implement a GAs:

1- an encoding
2- an initial population
3- an evaluation using a fitness function
4- a selection procedure
5- a crossover procedure
6- a mutation procedure
7- a set of stopping criteria.

Figure 3.8: General Procedure for a Genetic Algorithm

3.3.1.13 SA (Simulated Annealing)

Simulated annealing is a generalization of a Monte Carlo method for examining the equations of state and frozen states of n-body systems [63]. It is an iterative technique based on the physical process of annealing. The physical process of annealing is the thermal process of acquiring low energy crystalline states of a solid. In this process, if we increase temperature, the solid is melted. For example if we decrease the temperature slowly, the melted particles of the solid arrange

themselves locally. The solid will reach an optimal solution, if we decrease temperature sufficiently. If we want to adapt this technique to the SA grid scheduling heuristic, this optimal solution is an optimal task-machine mapping, that is, the optimization criteria. Therefore the total completion of a metatask, makespan, is the temperature and the process of mapping change is the change of temperature. For example; the next temperature being high means worse mapping, in this case the next state is accepted because these worse states can be used to escape local optimality [13].

In the SA heuristic, only one possible solution (mapping or scheduling) is considered for each metatask (application or job) at a time. If we want a better solution, the initial implementation of the SA is firstly evaluated and then modified and finally refined. The makespan of the initial mapping is the initial system temperature.

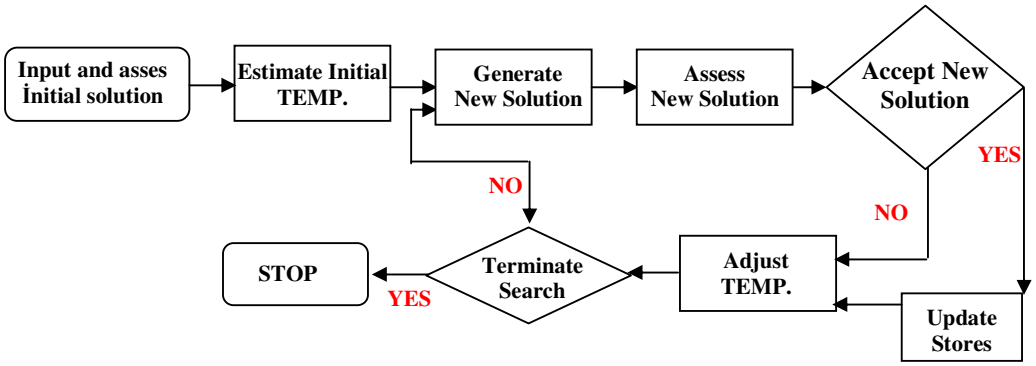

Figure 3.9: Iteration of SA [1]

The SA heuristic has two procedures: The initial and final procedures.

In the initial procedure, a uniform random distribution is used to generate the first mapping. Mutation procedure is applied on the mapping as the GA. After that the generated new makespan is evaluated. At that point, this new mapping changes the old one if and only if the new makespan is better. On the other hand, if the new makespan is worse, a uniform random number z $\in [0, 1]$ is selected and then z is compared with y, where

$$y = \frac{1}{1 + e^{\left(\frac{\text{old makespan - new makespan}}{\text{temperature}}\right)}}$$

if z<y then the new mapping is rejected, otherwise it is accepted (poorer) [1]. After that the algorithm keeps old mapping. Finally the system temperature is reduced to %90 of its current value after each mutation procedure. The iteration of heuristic is continued until the system temperature approaches zero or there is no change in the makespan, heuristic is stopped.

In [94], a scheduling in GrADS (Grid Application Development Software) using SA is an example of Simulated Annealing heuristic. The scheduler uses the Metacomputing Directory Service to get a list of the available machines, and uses the Network Weather Service to get CPU load, memory and communication details for the machine.

In [95], another variation of SA is used to choose how many resources a Grid application should be split over.

Since, in the SA heuristic, very poorer solutions are accepted in the initial levels and it does not correct this problem during the heuristic, GA (or Min-min) gives better solution than SA [1].

3.3.1.14 TS (Tabu Search)

It is a classical optimization method often encountered when faced with the difficulty of solving hard problems plentiful in the real world. Tabu search (TS) goes beyond the classical design to provide a method dramatically changing our ability to solve problems of practical significance [58].

We can define Tabu Search as a solution of (mapping) space search. It is an iterative technique. The solution space search traces the regions of the solution space that have already been searched so as not to repeat a search near these areas. As in GA heuristic, chromosomes are used as solution (mapping) representations. The initial solution is created by a uniform distribution. A short hop is defined by the Tabu Search to find the nearest local minimum solution in the solution space and to manipulate the current solution. Also it is used to move current solution through the solution space. A short hop is performed in a way that each possible pair of tasks and each possible pair of machines assignments are considered. While this is done, the other assignments are unchanged. This procedure is applied for every possible pair of tasks. The heuristic saves the new solution by replacing the current solution, if the new makespan is better than the old one. Whether every pair-wise remapping does not produce better makespan or successful hops reached limit, the short hop procedure is ended. When the short hop procedure ends, the final mapping is added to the tabu list. After that a long hop is applied. The aim of the long hop is to generate a new mapping that must differ from each mapping in the tabu list. The Tabu search heuristic repeats the short hop procedure after each successful long hop procedure. When both total number of short hop and long hop reach the limit, entire heuristic is stopped. Finally the best mapping in the tabu list is the final answer [58].

3.3.1.15 Hybrid GA with SA and TS

The hybrid mode heuristic can be generated as a combination of SA and TS with GA. This combination can be done in two ways. First combination is GA with SA and the second one is GA with TS.

The GA and SA combination is called The Genetic Simulated Annealing (GSA) [75], [26]. GSA very similar to GA heuristic but for the selection process, It chooses the SA rather than GA. In the selection process, to accept or reject a new chromosome, GSA uses system temperature, the SA cooling schedule, and a simplified SA decision process.

The other combination is GA and TS that is called GTS. The researches show that while TS provides very flexible, powerful and easy to implement, performance of TS mostly relies on the selection of parameters and qualified information of the problem required to be solved. Therefore combination of GA with TS makes the heuristic more robust than GA and TS. Reproduction, crossover, and mutation process that in the GA heuristic is replaced by reproduction, crossover, and Tabu Search process. Rather than mutation changes, in the hybrid GA-TS each member of the population goes through a come apart optimization process. This optimization process is described by a Tabu algorithm in [1].

3.3.1.16 HMM (Heterogeneous Multi-phase Mapping)

It is a static graph-based mapping algorithm called Heterogeneous Multi-phase Mapping (HMM). HMM is the combination of Tabu Search and meta-heuristic. By using these local search techniques, it allows a sub-optimal mapping of a parallel application on heterogeneous systems. HMM assigns parallel tasks by exploiting the information embedded in the parallelism forms used to implement an application [10].

3.3.1.17 LMT (The Levelized Min Time Algorithm)

The LMT is a combination of the Level Sorting heuristic and Min-Time heuristic. Actually it has two phases. The Level sorting is the first phase and Min-Time is second phase. In the Level sorting phase, all nodes are put in order on the basis of their precedence constraints. Tasks within each level have no precedence constraints between them as seen in the Figure 3.10, so tasks are clustered to able to execute in parallel. After that each task is assigned level by level using Min-Time heuristic. In Min-Time each task is mapped to the best machine that runs it fastest [49].

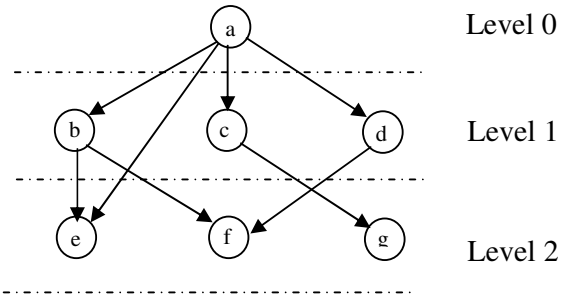

Figure 3.10: LMT heuristic

3.3.1.18 Cluster-M Mapping Heuristic

In the Cluster-M mapping heuristic, the mapping process (matching + scheduling) is defined in a way that it represents a set of subtasks (task graph) onto a graph and the set of machines in grid environment (system graph). Cluster-M Mapping heuristic has two phases; tasks clustering and machines clustering phase. In the first phase, the system and task graph are clustered separately. The aim of the task graph clustering is not only to combine communication intensive subtasks into

the same cluster but also to combine the machines that are tightly coupled (small inter-machine communication times) into the same cluster. The second phase is mapping. The heuristic maps the clustered task graph on a clustered system graph. The clustering improves the quality of mapping and decreases the complexity of the mapping problem. Moreover clustering is an efficient way to reduce communication delay in DAGs (Directed Acyclic Graph) because heavily communicating tasks can be grouped into to the same clusters [61].

3.3.1.19 K-Distributed Heuristic

In K-Distributed Model, there is a metascheduler responsible for all local scheduler. It is introduced by Subramani et al [81]. In this heuristic there are K distinct distributed sites. Initially, each job is sent to the K least loaded sites. K can be varied according to the required scalability. After that each of these K sites schedule the job locally. If a job is started at any of the sites, this site informs the other K-1 sites to remove the job from their respective queues. Therefore all jobs are cancelled. This improves the resources utilization and reduces makespan. This algorithm does not use performance prediction to make scheduling, whereas all algorithms we showed above use performance estimation. The idea behind the K-Distributed heuristic is duplication. [81].

3.3.1.20 WQR (Workqueue with Replication)

WQR is developed by Silvia et al [78]. It is based on classical workqueue algorithm. Workqueue is a knowledge-free scheduler that does not need any kind of information for task scheduling. In the workqueue heuristic, initially all tasks are grouped into a bag and then send to the available resource. Whenever a resource finished its task, another task is sent to the resource by the scheduler. However, the resource waits idle during this time interval. In the WQR heuristic, the heterogeneity of resources and tasks can be achieved using task replication.

Moreover task replication can be used to deal with the dynamic variation of resource availability because of load imbalances. In the classical workqueue algoritm, when the resource finishes the task asssigned to it, the resource waits idle. However in the WQR heuristic, all these idle resources are asssigned to the replicas of tasks that are still running. Tasks replication continues until the maximum number or replicas defined beforehand is achieved. If a resource finishes a task, other replicas of the task are cancelled. The advantage of this approach is that it does not need to use information of about speed and load of resources and length of tasks. However, when tasks replicas are canceled this causes wasted CPU cycle [78].

3.4 Dynamic Scheduling

In dynamic scheduling technique, also called on-line scheduling, some or all the decisions are done during the execution. It is suitable when jobs and machines are coming online, or going offline due to failures, the processor speed of each processor vary during the scheduling and difficulty in predicting the cost of applications. Moreover dynamic mapping is performed when the arrival of tasks is not known beforehand.

Since grid has changing resource availability, network performance, user requirements, and system workload, dynamic scheduling reacts on following situations:

- Resource or job failure
- Resource or job changes
- Critical job arrivals
- Performance contract violation

In brief, dynamic scheduling algorithms study effect of run-time changes in grid resource properties and user requirements on system performance. These situations are mostly job or grid related. In job situation, some job failures can occur, or a high priority job can arrive, or some changing can occur in job priorities and deadlines. On the other hand, in grid related, while some resource failure may occur, new resources may come online, or resource workload can change. These dynamic situations can cause inefficient resource usage which decreases the resource utilization. Moreover they can lead job failure and do not provide users' QoS requirements [27].

Dynamic scheduling algorithms can be divided into two subgroups, as shown in Figure 3.1.

- Distributed Dynamic Scheduling Algorithms
- Centralized Dynamic Scheduling Algorithms.

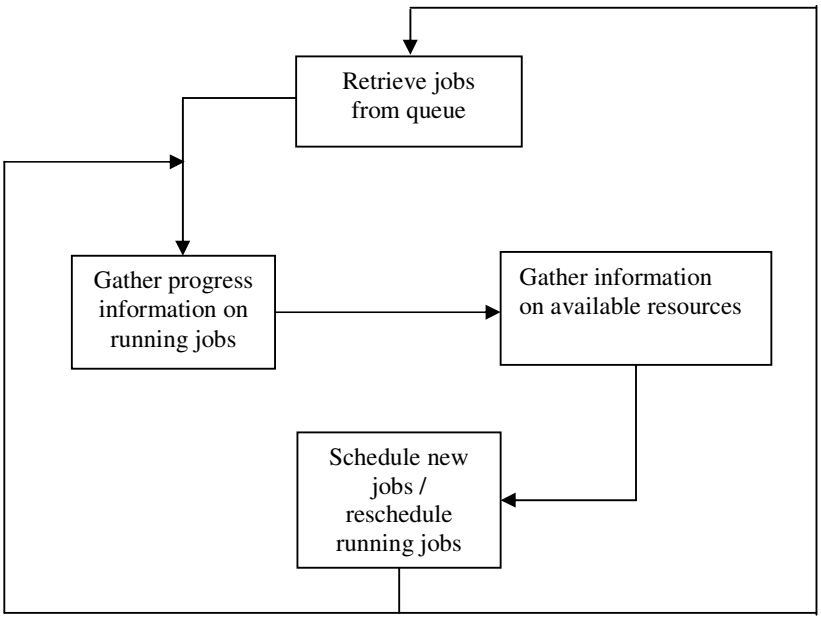

Figure 3.11: Generic Dynamic Scheduling Procedure

In dynamic case, scheduling decision may be done with one general centralized scheduler, or multiple distributed schedulers. While the centralized scheduling algorithms are implemented easily, they do not support up-to-date load balancing, fault tolerance, and scalability. Therefore this type of scheduling technique gives a performance bottleneck. In distributed model, there is no central leader responsible for scheduling. Therefore we do not encounter disadvantages of centralized scheduling model. Decentralized scheduling strategy is suitable for grid systems because resource owners can define their scheduling policies that schedulers can enforce at will. However, resource owners do not agree on a

common policy goal for resource management, so development of scheduling algorithms seems to be difficult [35].

We can investigate Distributed Dynamic Scheduling Algorithms in two subcategories.

- Cooperative
- Non-Cooperative

In the cooperative case, each scheduler has two goals. The first is acquiring its own system goal, and the second is to provide a common global goal with all the schedulers in the grid. That means there is a local policy for each local grid scheduler to achieve some global goal. However, in the non-cooperative case, each scheduler is responsible for their own goal (optimize their private individual objects) and does not consider the rest of the system. That means there is no a common global goal for the entire grid. Application level schedulers are good example of this type of grid scheduling.

3.4.1 Examples of Dynamic Scheduling Algorithms

3.4.1.1 DFPLTF (Dynamic Fastest Processor to Largest Task First)

It is based on static FPLTF (Fastest Processor to Largest Task First) heuristic introduced by Silva et al [78]. It is useful for BoT (Bag-of-Task) applications. BoT applications are composed of lots of tasks that are completely independent and do not need inter-task communication. DFPLTF heuristic assigns a task according to its priority is defined by the heuristic. The largest task is assigned the highest priority. Hence, the heuristic assigns the task with the highest priority to the fastest processor. However DFPLTF must need the knowledge about the speed of processors and length of tasks in advance [28].

3.4.1.2 Chang's Algorithm

It is suitable for soft real-time systems that decline the value of an operation steadily after the deadline expires. That means tasks are completed after their respective deadlines are less important than those whose deadlines have not yet expired. This leads to some tasks not being performed, if the system load is too high. This algorithm has a two phase polling strategy. It is based on Shortest Processing Time First local scheduling heuristic. A late job may give harmful outcomes in the hard real-time systems. A soft real-time system functions correctly as long as the deadline miss ration and the expected lateness are below pre-defined levels [25].

3.4.1.3 Shin's Heuristic

It is also called Load Sharing Method with State-Change Broadcast (LSMSCB) [74]. It is a load sharing method for distributed real-time systems. Each node in the systems maintains state information in the small set of nodes in its physical proximity which is called a buddy set. The three threshold values are defined as the load state of the node and QL is defined as the queue length. TH_u, TH_f, and TH_v are these three thresholds.

In this algorithm, each node must maintain and update the state information of other nodes. An overloaded node can transfer a task to another node based on state information without any probing delay.

For instance;

If a node's QL is less than or equal to TH_u, $QL \leq TH_u$, the node is said to be underloaded. If it's QL greater than TH_u and less than or equal to TH_f, the node is said to be fully loaded. And finally, QL is greater than TH_f and less than or equal

TH_v, the node is said to be overloaded. If new arrived job and/or completion of tasks make the node fully loaded, the node broadcasts its change of the state to all the other nodes in its buddy set. After that every node in the buddy set that receives this information updates its ordered list (preferred list) of available receivers. If there is an overloaded node in the buddy set, it can select the first underloaded node in its ordered list and then send a task to selected node. This cycle continues until all tasks are finished. This heuristic can complete the tasks before their deadlines [74].

3.4.1.4 Ramamritham's Heuristic

It is a distributed scheduling heuristic that schedules tasks according to their deadlines and resource requirements. This heuristic uses both local and global scheduling approaches in distributes systems. It is used for both periodic and aperiodic tasks. Local scheduling techniques are used for periodic tasks and global scheduling techniques are used for aperiodic tasks. In the local scheduling, all periodic tasks are known in advance and can always be scheduled locally. On the other hand in the global scheduling, an aperiodic task may arrive at a node at any time and if its deadline can be met this node, it will be scheduled locally on the node, if not; task will be transferred to remote node [70].

Four algorithms were used to select a remote node for each aperiodic task.

The random scheduling algorithm: The aperiodic task is sent to a remote node randomly.

The focused addressing algorithm: The aperiodic task is sent to a remote node that is estimated to have sufficient excess to complete the task before its deadline.

The bidding algorithm: The aperiodic task is sent to a remote node based on the bids received for the task from remote nodes in the system.

The flexible algorithm: The aperiodic task is sent to a remote node based on a technique that combines bidding and focused addressing [70].

3.4.1.5 Zhang's Heuristic

Zhang proposed a general three level dynamic scheduling algorithm [97]. It is based on the Shin's heuristic [74]. In this model, the grid has M clusters connected by the Internet and each cluster is composed of Ni nodes. To collect the information of the grid, a set of thresholds and information lists are used. Six thresholds values are defined. Three of them CTH_u, CTH_f, and CTH_v are used for cluster and the others NTH_u, NTH_f, and NTH_v for nodes. Therefore according to these values;

There are four states for each cluster and each node in the grid.

A cluster is said to be underloaded if the number of tasks in it, N_c, is less than or equal to CTH_u

$$N_c \leq CTH_u$$

Is medium loaded if $CTH_u < N_c \leq CTH_f$

Is fully loaded if $CTH_f < N_c \leq CTH_v$

Is overloaded if $N_c > CTH_v$

A node is said to be underloaded if if number of tasks in it, N_n, is less than or equal NTH_u

$$N_n \leq NTH_u$$

Is medium loaded if $NTH_u < N_n \leq NTH_f$

Is fully loaded if $NTH_f < N_n \leq NTH_v$

Is overloaded if $N_n > NTH_v$

In this heuristic, there is one server for each cluster. Both the state information of other cluster and the state information of each node of this cluster are collected by that server. Heuristic is working like that: If a node gets fully loaded, its change of state will be sent to the server (each cluster has a server) and then the server updates its state information table. If any change occurs in the state of cluster e.g. from fully loaded to the overloaded, it will send this alteration to the all other clusters in its buddy set. After that every cluster which receives this information updates its ordered list (preferred list) of available receivers. This heuristic is applied to three levels. In the first level also called node scheduling, when a task arrive at the node, the scheduler acquire the required information about the task and node's state and then uses these information to decide whether the execution of the task is guaranteed on this node. If time constraint of task can be met, this task is scheduled on this node. If not, the cluster scheduling (the second level) is started. In this level, the sate information about the cluster that too near the node is used to decide whether the new task is accepted or not and its time constraint is provided by this cluster. After that if the new task is accepted by this cluster, the scheduler transfer this task to one underloaded node in this cluster. The task execution is done on this node. Finally if the cluster does not accept the new task, third level scheduling is started. It is called grid scheduling. The scheduler find a remote cluster in the grid to accept new task by using information list and then new task is executed on that remote cluster [97].

3.4.1.6 Gu's Heuristic

While most scheduling algorithms do not take account performance issues, the aim of the Gu's heuristic is adopting the performance of the grid system by performance evaluation and optimization. It is the grid workflow based scheduling heuristic. The performance arguments are added to heuristic for the resource management and to evaluate the execution performance of grid service in order to facilitate next scheduling.

A set of activities (grid service) $A_i(i=1,\cdots,n)$ defines a grid workflow, where n is the sum of the number of activities. Resource can be defined as a list of resource class $R_k(k=1,\cdots,m)$, where m is the sum of resource class. And k composed of k_j $(k=1,\cdots,j)$ that is containing same type of resources R. Moreover each grid services consist of one or more resources. The engine schedule, not only matches performance arguments between the resources and grid service, but also selects the required type of resources. The typical features of the heuristic are like as follows:

1. The engine is multi-thread. Hence it schedules many instances simultaneously.
2. If the workflow model can be added QoS, the engine schedules the resources according the QoS and performance.
3. While the algorithm omits the detail of QoS, but focuses on the performance issues [45].

3.4.1.7 MECT (Minimum Execution Completion Time)

It can be thought as the combination of MET and MCT.

e_{ij} → execution time,

b_{ij} → begin time of task t_i on machine m_j

c_{ij} → completion time of task t_i on machine m_j

The completion time of a task t_i on machine m_j is computed as follows:

$$c_{ij} = b_{ij} + e_{ij}$$

In MECT heuristic, the inputs consist of execution time of each task on all machines. That is

$$T_i = \{e_{i1}, e_{i2} \ldots e_{im}\}$$

MECT firstly finds b_{max} which is maximum begin time between the all machines' begin time. And then it determines M' that is a subset of machines which has all e_{ij} like $e_{ij} < b_{max}$. Thirdly, if there are e_{ij}s like that M' is not empty, MECT finds a machine Mk with minimum execution time to execute the task t_i. If M' is empty means that there is no e_{ij} like that, MECT finds the machine M_k in M with minimum completion time to execute the task t_i and then index of machine k is returned by the MECT [53].

According to [60] dynamic scheduling algorithms can be grouped into two sub-groups.

- Batch Mode
- Immediate (on-line) mode

While batch mode heuristic considers a subset of tasks, immediate-mode (online-mode) heuristics only consider newly arrived tasks. Studies show that batch mode heuristics give higher performance than immediate-mode heuristic. On the other hand, immediate-mode heuristics provide shorter running times than batch-mode heuristic.

3.4.1.8 Max-Max

It is based on greedy concept. In this heuristic, all computation is done on the basis of fitness value of the task. The fitness value of a task can be computed as the value of the task divided by the estimated execution of the task. On the other hand the value of the task can be computed by the prior weight of the task multiplied by the deadline factor of the task. Fitness value is used to select the task that being mapped. Moreover the fitness value calculates the worth per unit time.

When a new task arrives and generates a mapping event, the Max-Max heuristic starts. The new task and tasks that are waiting to be executed in the

machine queue are called mappable task. The heuristic can be examined by the following six steps.

1- All the mappable tasks are defined by generating a task list.
2- Find the machine that gives the task with the maximum fitness value, for each task in the task list, (this is the first “Max” in the Max-Max.) and pair this task and machine.
3- Find the task/machine pair with maximum fitness value (this is the second “Max”) among the all task/machine pairs.
4- Founded task with maximum fitness value is removed from mappable task list and assigned to its paired machine.
5- Machine available status is updated
6- Steps 2 to 5 are repeated until all tasks are mapped.

After every task mapping, the worth values for all task on all machines are recalculated. Moreover in step 5, to calculate the deadline factor, the availability status of all machines is updated. The availability time of the machine plus the estimated execution time of the task gives us the deadline factor for a given task/machine pairs [54].

3.4.1.9 Max-Min

The completion time for task i on machine j is calculated as the machine availability time (mat(j)) plus ETC(i, j). Max-min heuristic finds the machine that gives the minimum completion time for each task. Then, from these task/machine pairs, a pair with the maximum completion time is selected by the heuristic. When a new task arrives and generates a mapping event, the Max-Min heuristic starts. Initially all machines’ queues are empty.

The heuristic can be examined by the following six steps [54].

1- All the mappable tasks are defined by generating a task list.
2- Find the machine that gives the task with the minimum completion time for each task in the task list (this is “Min”) and pair this task with machine.
3- Find the task/machine pair that gives the maximum completion time (this is “Max”) among all task/machine pairs.
4- The found task with maximum completion time is removed from mappable task list and assigned to its paired machine.
5- Machine available status is updated
6- Steps 2 to 5 are repeated until all tasks are mapped
7- Reschedule the tasks in the machine queue according to their worth, if there are tasks in the machine queue. Do this for each machine.
8- If all machines queues are rescheduled, stop the algorithm.

In step 5, the availability status of all machines is updated to calculate the minimum completion time over all machines for each task in step 2.

The rescheduling of tasks in step 7 can be explained as in the following six steps:

1- Initialize the machine availability time to the completion time of the very next task that is waiting to be executed
2- The tasks are grouped by using the priority levels of the tasks.
3- For the tasks in the high priority level group, keeping their relative ordering from the machine queue, one by one, in order, insert the tasks that can finish by their primary deadline into the machine queue. Whenever scheduling is done, heuristic removes a task from the group and updates the machine availability status.
4- Step 3 is repeated for 50% deadline and 25% deadline.
5- Steps 3 to 4 are repeated for the medium priority tasks and then repeated for the low priority tasks.

6- If high priority tasks cannot finish by the 25% deadline, they are added to end of the machine queue. On the other hand, if medium priority tasks cannot finish by the 25% deadline, they are added next and then the low priority tasks are added to the machine queue.

Min-min heuristic is very similar to the Max-min except that in step 3, instead of selecting pair that gives the maximum completion time, the pair that gives the minimum completion time is selected [54].

3.4.1.10 Percent Best

It is based on KPB (k-Percent Best) heuristic. This heuristic not only considers the tasks completion times on the machines but also tries to map the tasks onto the machine with minimum execution time. The goal of the heuristic is to collect the top m machines with the best execution time for a task so that the task can be assigned to one of its best execution time machines. However in this heuristic, system may become unbalanced because of limiting the number of machines that a task can be mapped. To prevent this, the completion time of the tasks should be considered by the heuristic [54].

When a new task arrives and generates a mapping event, the Max-Min heuristic starts.

The heuristic can be summarized by the following nine steps [54].

1- All the mappable tasks are defined by generating a task list.
2- Heuristic groups the tasks according to their priority level.
3- In the high priority level, the top m machines with the best execution time are found for each task.

4- Machines with the minimum completion time in step 3 and the machines that are idle are found.
5- Tasks are mapped with no contention that is there are no other tasks with the same minimum completion time machine. After that mapped tasks are removed from the priority level group.
6- On the other hand, if there is a task with contention, the task with the earliest primary deadline is mapped and then removed from the priority level group.
7- Machine availability status is updated.
8- Steps 3 to 7 are repeated until all tasks in the group are assigned.
9- Steps 3 to 8 are also repeated for tasks in the medium and low priority level groups.

3.4.1.11 Queuing Table

This heuristic maps a task to a resource by using a lookup table. This lookup table is constructed by using priority level, the relative speed of execution, and the nearness of deadline (NOD). The ratio of the average execution time of a task across all machines to the overall average task execution time for all tasks across all machines in the heterogeneous environment is called the relative speed of execution (RSE) [54].

In this heuristic, tasks are classified into one of two categories: “slow” and “fast”. To do this, the heuristic uses the above definition and RSE-cutoff is called constant.

The heuristic can be summarized by the following nine steps [60].

1- The NOD is calculated for all mappable tasks.
2- RSE is calculated for new task

3- The new task is compared with the tasks in the machine's queue. If the machine's queue has not a task with the similar queuing order to the new task, the new task's position is in front of the first task with the higher numbered queuing order. If the machine's queue has tasks that it's queuing order same as the new task, the new task's position is in front of the first task that has a higher NOD value than that of the new task. This is done for each machine.
4- The completion time on all machines is calculated by using the position on each machine. The position is also used to map the new task to machine with minimum completion time.
5- The heuristic check whether any tasks will miss their primary deadline for each machine.
6- Find the first task that misses its deadline among all the tasks that missed its primary deadline.
7- A machine is found for the task is found in step 6, where
    a- The task' priority is equal to or greater than the any task with the highest priority on that machine and
    b- The task is moved to the front of that machine queue without causing any task to miss its primary deadline
8- The machine with the minimum completion time is found and the task is moved to the head otherwise the task is not moved without the machine in step 7.
9- Machine availability status is updated
10- Steps 5 to 9 are repeated until all machines are checked.

3.4.1.12 Relative Cost

It uses combination of worth and suffrage ideas to assign the tasks. The heuristic calculates the relative cost (RC) by computing the minimum completion time of that task over all machines and then divides this value with the average

completion time of that task on all machines. This calculation is done for each mappable task. If RC is low, the minimum completion time is too different from the average completion time. Otherwise, the minimum completion time is similar to the average completion time and we can say most of the completion times on all machines are similar. For example; the best machine m is preferred by tasks x and y for mapping. If it is assumed that there is a larger difference between the completion times of the best and the second best machines, task x is to suffer more than task y. The RC is an approximation of this difference. We can say a task will suffer less than a task that has a low RC, when the RC for a task is high [54].

3.4.1.13 The History Based Approach for Run-Time Estimation

Run time estimation based on history has been developed by Shonali et all [57]. It uses run time estimation is based upon Rough Sets theory [56], [57], and [68]. In this approach, authors aim that if two applications are similar to their characteristics, then their runtimes are also similar. To do this, a history of applications that have already executed is recorded and maintained. These characteristics are called applications' attributes. They are classified into two sub groups: Condition and Decision attributes. Condition attributes represent the applications characteristic like length, input and output size etc. On the other hand, Decision attribute is the runtime of the applications. There is a strong relationship between the decision attribute and condition attributes. Indeed, condition attributes play an important and active role while defining the applications runtime.

Moreover, according to Aggarwal et all [4] there are two methods to store the history information of a job execution: Centralized history for all site, and decentralized history at each site. The difference between these two methods is the number of history. In the centralized case, the information is stored in a central job history database. In the decentralized case, each execution sites its own job history. Their approach is also based on Rough Set theory. They developed a prediction

engine works as a part of Grid scheduler. This prediction engine will provide estimates of the resources required by job submission based upon historical information.

CHAPTER 4

STUDY OF MIDDLEWARE AND SIMULATION TOOLS FOR GRID COMPUTING

4.1 Introduction

Grid computing, most simply stated, is distributed computing taken to the next evolutionary level. The goal is to create the illusion of a simple yet large and powerful self managing virtual computer out of a large collection of connected heterogeneous systems sharing various combinations of resources.

There are many factors to consider in grid-enabling an application. One must understand that not all applications can be transformed to run in parallel on a grid and achieve scalability. Furthermore, there are no practical tools for transforming, arbitrary applications to exploit the parallel capabilities of a grid. There are some practical tools that skilled application designers can use to write a parallel grid application. However, automatic transformation of applications can be a difficult task.

In some cases, the burden of creating such environment might outpace the effort required to enhance the capabilities of such systems. In the following section 4.2, a brief introduction to few well known grid tools will be given for completeness Globus, GridBus, and NetSolve/GridSolve. However, no grid environment will be created. Instead a simulation tool will be used to concentrate on the particular history injected scheduling concept will be discussed in Chapter 4.3.8.

In Chapter 4.3, a survey on simulation tools that simulate grid environments is presented, which provide enough information to decide on the simulation tool best studied to our algorithm. The simulation tools discussed are OptorSim, ChicSim (The Chicago Grid Simulator), EDGSim (European Data Grid Simulator), GridNet, Bricks, SimGrid, MicroGrid, GridSim. From these, we have chosen GridSim for testing the performance of HISA (History Injected Scheduling Algorithm) algorithm.

4.2 Study of Grid Middleware for Grid Computing

In order to build Grid systems and applications, many toolkits are developed by different commercial or non-commercial companies. Globus, GridBus, and NetSolve/GridSolve are some of such toolkits.

4.2.1 Globus

Globus is an open source toolkit allows people share their databases, applications, computational power and others tools over the Internet in terms of secure and seamless manner. The Globus can be thought as a middleware that includes software services and libraries for resource monitoring, discovery, and management, and also security and file management. It is being developed by the Globus Alliance which is an active member in the community of Grid Software developers and many others all over the world [87]. It is a portable so it can be used

for any platform. Resource and data management, authentication and authorization, security, communication, and fault detection are supported by the Globus. It also provides Web services mechanisms to create distributed system framework. Any local scheduling mechanism (e.g. Condor, LSF, PBS etc.) can easily be used with Globus. Globus can be run on any operating systems.

4.2.2 GridBus

It is also an open source toolkit. GridBus is developed by Grid Computing and Distributed Systems (GRIDS) Laboratory at University of Melbourne [88]. It is composed of middleware, tools, and applications. GridBus provides economic grid scheduler, cluster type scheduler, grid modeling and simulation tool, Grid Bank, and GUI for workflow management. Actually it is mainly focused on Grid Economy model. For example; while Libra is used for economy based scheduler for clusters, GridSim is used for modeling and simulation of Global Grids. Moreover, in economic model Grid, GridBank is used for grid accounting, authentication, and payment management. GridBus provides service broker for scheduling distributed data oriented applications across different types of grid resources [88].

4.2.3 NetSolve/GridSolve

It is a client/agent/server system based on GridRPC. GridRPC is based on traditional Remote Procedure Call (RPC) programming paradigm. The purpose of NetSolve/GridSolve is to create a Grid middleware between the computational resources in the Grid environments.

NetSolve/GridSolve has four main components: NetSolve Client, Servers, Agent, and Monitoring. Client is a library and is used with the applications of users to call NetSolve's API. Currently NetSolve client library supports C, FORTRAN, Matlab, Mathmatica, and Ocatave. NetSolve server responds the clients' requests.

With the server any type of architectures from single PC to symmetric multiprocessor or machines with massively parallel processor can be run. NetSolve agent acts as a Resource Broker. It has a scheduler responsible for job assignment. It has also a database containing the resource information. This information can be load on servers, available bandwidth, current load on the resources, installed software etc. All resources in the Grid register themselves to the agent and then the agent chooses the best resource, sends the jobs, and finally collects the results. Monitoring service is used with NWS (Network Weather Service) to monitor the CPU and network connections load. This provides the most sophisticate information and forecasts of both resource and network status and availability [67].

4.3 Study of Simulation Tools for Grid Computing

To prove the effectiveness of the scheduling algorithms, and their performance, one needs to evaluate, and then use different scenarios with varying number of resources and users with different QoS requirements. To build a Grid test bed is expensive and time consuming. In such a test bed, investigating and testing resource management and scheduling strategies for Grid computing are too difficult and also impossible to trace. Moreover, execution of applications is continuing for hours, days, and months, that is, testing scheduling algorithm can be time-intensive. Finally experiments are limited to the test bed. Therefore we chose our test bed to be a simulating Grid environment.

4.3.1 OptorSim

It is time-based simulation systems that used for Grid simulation. Actually OptorSim focuses on data replication. With OptorSim various replica optimization algorithms can be evaluated. Computing and Storage Elements can be modeled according to real computing and storage resources. Job scheduling is controlled by a Resource Broker. Each site has a Replica Optimizer used to create and delete

replicas. Currently OptorSim is used by UK Grid for Particle Physics (GridPP) and European DataGrid (EDG) projects for replica management [18].

4.3.2 ChicSim (The Chicago Grid Simulator)

It is a discrete-event data location-based grid simulator. It is built on Parsec that is a simulation language based on C language [9] [86]. ChicSim can be optimized across multiple tasks and it is easily adapted to the changes in the Grid environment. The primary of aim of the ChicSim is to evaluate replication strategies. It supports a framework to implement a wide variety of scheduling and replication algorithms and also evaluates the effectiveness of these different approaches/algorithms for scheduling. The ChicSim models a Grid as a collection of sites and a certain number of processors of equal capacity and limited storage that is identical in each site. Before a job starts, certain data-files are required to present it locally. Also the job's completion should be on a single processor. In ChicSim, job scheduling algorithms run a job at a randomly chosen site, at least loaded site, or where input data that is required for the job is, already available. On the other hand, data scheduling replicates popular data files at random site or at the least loaded neighbor [71], [69].

4.3.3 EDGSim (European Data Grid Simulator)

It is a simulation of the European Data Grid project [33]. The aim of the EDGSim is to replicate the communication between the components of the EDG, instead of reproducing their complex functionality in exact detail. It is a discrete-event simulator and developed with Ptolemy II java-based object oriented software. Ptolemy II application consists of objects known as Actors representing nodes in the grid. They are managed by a Director object that checks both the system parameters and the event queue. The Actors communicate by exchanging Data Tokens which are wrapper objects containing anything from an integer to a complex object in its

own right [34]. This is necessary because in the simulated system, any change in state (e.g. a job being generated, a file transfer beginning, and an information update) is recorded as a time-stamped event in a queue, and then the events are coped with in chronological order. In the EDGSim, after a job is created, firstly it is scheduled, secondly submitted, thirdly queued, finally run and completed. Each site has Storage Element (SE) and Compute Element (CE) that manages the worker machine. Also there is a Replica Catalog and a Resource Broker in each site. Initially, the job is sent to the Resource Broker and then it passes the logical names and required data files of job to the Replica Catalog. The Resource Broker assigns the job to a free machine at the CE. If there is no available machine, the job is queued until one becomes available. The job can only begin to run, whenever at least one of the required data files is present at the local SE. If the data files are not present, the SE may attempt to get them. Whenever the job is finished, it is removed from the system. The results of the run with information about the performance of individual jobs as well as the system as a whole are output to a plain text file. This information is sent to other EDG component in the systems. This provides coordinating the transfer of large data files required to run a job, and the jobs are dispensed effectively between available machines [33].

4.3.4 GridNet

It is a modular replica simulator based on NS (network simulator) [24] and written with C++. NS provides generation of different network topologies. GridNet's applications level services are implemented on top of existing NS protocols. The main goal of GridNet is to create a certain number of replicas on a given node to guarantee minimal availability requirements. In order to provide this, different replication and caching strategies are developed. With GridNet different Data Grid configurations, resources, nodes, links, and messages can be modeled.

Each node in the Grid can manage its storage capacity, organizes its local data files, specifies its neighbors, and peer replica nodes.

In GridNet, Grid user requests are represented by the packets. These packets are also used to represent the start and the end of grid data transmission in NS that transfer control of the simulation to GridNet. The latter simulates the replication decision at each node and generates new NS traffic that forwards request from one node to the other or sends the requested data to the client.

GridNet has a replication service. The mission of the replication service is to start data replication when needed. Moreover, for each data request, multiple replicas are created on multiple nodes so this provides the shortest access time to the requested data. In order to access existing replica, by creating a new one or deleting some of them, the replication service can be used.

4.3.5 Bricks

It is a performance evaluating system that work as a discrete-event simulator of a queuing system. The Bricks is written in Java. It is developed by the Tokyo Institute if Technology [85]. With Bricks, one can analyze and compare various scheduling heuristic on a Grid system. It focuses on comparison and evaluation of different scheduling heuristics. Network topologies, resource architecture, communication models, and scheduling strategy can be modeled by the user by using Bricks scripts. Therefore Bricks provides easy creation of any type of Grid systems. In Bricks, there is a scheduling unit composed of scheduler, network monitor, predictor (network and server), network and server monitor. With Bricks, one can easily monitor, predict and schedule the task by using its defaults components such as Bricks network monitor and server monitor. Actually it supports centralized global scheduling method so there is a central resource broker that is responsible for all systems. On the other hand, in our simulation model, each

user has its own resource broker that manages application scheduling. That means we used decentralized scheduling model, in which there is no central scheduler. Therefore Bricks is not suitable for our simulation model [85].

4.3.6 SimGrid

It is a C language based simulation toolkit to study and evaluate the scheduling algorithms for distributed and Grid systems. With SimGrid, resources and tasks can be modeled via provided C APIs. Latency and service rate are two features that define the resource performance characteristic. In the SimGrid, resources are modeled as time-shared systems and are created in terms of a standard machine capability and also tasks are created in terms of their execution time. Simulation of multiple competing users, applications, and schedulers and recovery from the resource failures with SimGrid is difficult. The current version of SimGrid does not provide evaluation of scheduling algorithms [22].

4.3.7 MicroGrid

It is developed by University of California at San Diego. MicroGrid provides systematic design and evaluation of middleware, applications, and network services for the heterogeneous Computational Grid. Scientific and recurring experimentation will be supported by these tools. MicroGrid is used with Globus to provide the illusion of virtual Grid. Heterogeneous structure in the virtual Grid can be modeled using a global virtual time modeling. Basic resource simulation models for computing resources, memories and networking are provided by MicroGrid. Also MicroGrid is able to explore a wide range of resource (network, compute, storage) environments, dynamic competitive loads [80].

4.3.8 GridSim

It is a discrete-event simulation tool and written in Java. With GridSim different class of schedulers, resource brokers, heterogeneous resources, applications, users, and networks can be simulated. Moreover information service for resource discovery interfaces for assigning application tasks to resources, and monitoring and managing their execution are also provided by the GridSim. All these properties can be used to simulate resource brokers (like Condor/G, Nimrod etc.) or Grid schedulers for evaluating performance of scheduling algorithms or heuristics [14]. It is also suitable for single or multiple administrative domains and distributed computing systems such as P2Ps, clusters, and Grids.

The some important features of GridSim are as follows:

- Heterogeneous types of resources can be modeled as operating under space-shared and time-shared mode.
- GridSim support time zone feature so resources can be located in any time zone.
- Each resource capability can be defined with respect to the standard machine, MIPS (Million Instructions per Second) or SPEC (Standard Performance Evaluation Corporation) benchmark.
- According to owners requests', holidays (weekends and any formal and religion holidays) can be mapped relying on resources' local time to model local (non-Grid) workload.
- Various application models like parallel application model can be simulated and application jobs without limitations can be mapped to a resource. These application jobs can be heterogeneous.
- Various tasks from multiple user entities can be assigned to a resource simultaneously.
- Networking can be simulated with GridSim. For example network speed between resources.

- With GridSim static and dynamic schedulers and scheduling policies can be simulated.
- For statistical purposes all operations or only selected operations can be recorded and analyzed by the GridSim statistical analysis methods. [14].

Among these simulation tools we choosed the GridSim because it provides us some powerful features while creating of our grid simulation environment. First of all, because it is a JAVA based, we can use it under any operating system. In this study, we attempt to simulate a computational grid for our university METU (Middle East Technical University). Because all resources used for specific jobs by the owner in METU, with GridSim he/she may want to use his/her machine only for specific purpose. Therefore, in the GridSim, a resource can be modeled according to user requests. With GridSim anyone can easily simulate a grid environment because creating of application model, tasks, resources, and network model is too simple. We can choose any type of application model for developing scheduling algorithms. Moreover, any type of scheduling heuristic can be easily embedded in GridSim. GridSim provides statistical analysis model, this feature is important for us since we develop a history based scheduling algorithm; we used this statistical information for testing our scheduling algorithm. Also GridSim is coming with some scheduling algorithms so we can compare our scheduling algorithm with these scheduling algorithms to measure the efficiency of our algorithm.

CHAPTER 5

SCHEDULING IN GRID COMPUTING USING HISA (HISTORY INJECTED SCHEDULING ALGORITHM)

5.1 Introduction

In this work, in order to test and evaluate HISA, we need to create a Grid Scheduling Framework. Through this Chapter, our grid scheduling framework will be explained in detailed. In this framework, firstly we created our Resource Model. In the resource model, we created nine resources with different characteristic that are available in our department and Computer Center at METU (Middle East Technical University). Each single resource was modeled after its real world counter part by using GridSim. After that we selected an application model for suitable our framework and scheduling policy. And finally we decided a scheduling policy that will be used in scheduling algorithm.

5.2 Simulation Environment for METU Computational Grid

Our complete Grid scheduling framework for METU is composed of three parts:

1- Resource Model
2- Application Model
3- Scheduling Policy

5.2.1 Resource Model

The aim of the Grid resource model is defining the characteristics of Grid resources. A resource is a space/time shared entity which may differ from each other in many aspects. It may be dedicated to a Grid, a PC that volunteers to the Grid or any other.

The description of a Grid resource can be either fine-grained or coarse-grained. A coarse-grained can only define the OS type and hardware architecture. On the other hand, a fine-grained resource can define all detailed information related to the resource. These are CPU speed, the number of CPUs, memory size, parallelism they supported and so on.

Our Grid model has a number of sites composed of computational hosts. All sites connect together with a WAN and each of which has its own local users with their resources and policies. In this model, there are no dedicated resources which mean they are used for both local and grid applications. Neither resource failures nor communication costs are considered in our study.

As we design computational Grid, we consider a computational resource and its capability, measured by its CPU speed. We can create any type of Grid Resource such as a machine with a single processor, distributed memory cluster of computers, or SMP (Shared Memory Multiprocessor). A resource can be managed by time-shared or space-shared operating systems. These types of operating systems use a round-robin scheduling policy for multitasking. On the other hand, cluster type Grid resource can be managed by space-shared schedulers that use queuing systems.

Moreover with space-shared systems, First-Come-First-Serve (FCFS), back filling, and Shortest-Job-First-Served (SJFS) resource allocation policies can be used. While in time-shared resources, one PE can be assigned more than one tasks, that is, a PE can be shared by several tasks, in space-shared resources, one PE can be assigned only one task at a time [14].

We created nine resources that are time and space shared with different characteristic. Each resource can be modeled with resource name, number of machines and MIPS rating of each machines, baud rate, current peak load, off-peak load, and holiday peak load. The resource's architecture (e.g. Intel Pentium, Sun Ultra, SGI origin, Compaq Alpha Server, AMD Turion etc.), operating system (e.g. IBM Irix, Linux, Windows, Solaris, OSF1 etc.), cost in terms of Grid $ / Application, time zone (GMT) that show its location, and application policy (time/space-shared) can also be defined by the user.

5.2.2 Application Model

An application model is used to define the characteristic of applications that will be used in Grid environment.

In a Grid environment, fine-grained application form is used to process the jobs. This means each job is sent one at a time by the resource broker to the Grid resource. After that each job processed one by one, and then sent back to the user again individually. Therefore these processes (sending, processing, and receiving the jobs one at a time) increment the total amount of time that is required to execute all the jobs from a user. The time for transmitting each job to the Grid resource plus the overhead processing time for each job at the Grid resource gives us the total execution time. If we send a small job that requires low processing capabilities to a resource with a very high processing capability, this will lead to poor utilization of the resource.

An application model can be in task-farming, process parallelism, DAG (Directed Acyclic Graph), divide and conquer, pipelining and ring-based, speculative parallelism etc., as described in [77]. For example in a coarse-grained form [65], small jobs of the users are grouped together according to a particular Grid resource processing capability by the scheduling policy and then these grouped jobs are sent to this resource. Moreover all job groups are submitted and received to/from the Grid resource by the scheduler (resource broker).

In this thesis, we used bag of task (BoT) applications model. BoT applications are those applications composed of various tasks that are independent of each other. That means the independent tasks can be executed in any order and there is no inter-task communication between them. Because of this, BoT applications are the most suitable application model for current Grid environments rather than tightly-coupled parallel applications that may easily cause a bottleneck [29].

We can use BoT applications in a variety of areas because of its simplicity and usefulness. For example, currently BoT applications are used by SETI@home [6], [73], Nimrod-G [15], and MyGrid [32] to schedule compute-intensive or parameter-sweep applications to resources. Moreover BoT applications can also be used in data mining practices, parameter sweeps, simulations, fractal calculations, computer imaging, and computational biology.

While defining the jobs that are independent in the GridSim, we can use **Gridlet** object which is a package that contains all required information about a job. This information includes the number of Gridlets, length (defined in MIPS), disk I/O operations, the size of the input and output files, and the job owner or originator. Thus, the properties of a job are packaged as a Gridlet. To determine execution time, the time required to transport input and output files between users and resources, and the time required to send back the results to the originator (job's

owner) are added up. In GridSim, the job length is defined in terms of time that it takes to run on a standard resource PE with a SPEC/MIPS rating of 100. For example, for a Gridlet that requires many Processing Elements (PEs) or CPUs, the Gridlet length is calculated only for 1 PE for simplicity.

5.3 Scheduling Policy

In this work, we will take advantage of History Based Approach for finding the most suitable resource and run time estimation for a task.

5.3.1 History Injected Job Scheduling

In this thesis, we propose a scheduling algorithm based on historical information that provides us the estimation of the resources required by the job submission. This scheduling algorithm has ability to find the most suitable resource. For this purpose, we develop an interface acting as a history management engine that collects resources load information on all sites by using monitoring service GIS (Grid Information Service), and also uses SimReport entity to record all information about Gridlet during the simulation. History management engine maintains execution history file. This history information consists of former Gridlets id or name; length, input and output file size, owner, deadline and budget values, resource consumption, run times, and cost. This information used to select the most suitable and optimum resource for a future Gridlet according to the scheduling policy HISA.

5.3.2 The History Based Approach

The use of history for run time estimation is inspired by the past history of the applications and the systems. In system case, the behavior of the system observed or recorded in the past for a certain mix of applications, type of

applications, input output patterns, and/or time based pattern will be used to estimate the present applications runs. In case of applications, the behavior of application run observed under certain system characteristics such as processing and memory capacities, system load, mix of load, etc can be used to estimate the present application. One of the problems in such case is to know if the present application has a recorded history. The similarity concept in finding if an application has a similarity with a recorded application in the past can be used. A history based approach for run time estimation has been introduced by Shonali et all [57]. This based on the idea that if two applications are similar, then their runtimes are also similar. There are various views for definition of similarity. For example we can say that two applications are similar, if their lengths are the same, or they have the same input or output file size. Moreover, we can say two applications are similar since the same user on the same machine submitted them or since they are required to operate on the same-sized data.

Centralized and decentralized job history database are two ways to store the history information from job. In the central approach, there is a single history database for the grid environment maintained by the prediction engine. When a job is arrived to the scheduler, the scheduler immediately passes it to the prediction engine. The centralized job history database is analyzed by the prediction engine to identify similar job. Then the prediction engine calculates the estimated run time by using applications attributes of the similar jobs found in the history. In the decentralized approach, each site (resource) has a history of jobs that were executed on that site. When a job is arrived to the scheduler for scheduling, the scheduler immediately passes it to the prediction engine which will in turn passes to a broker at the job submission site. In this study, HISA is based on a centralized history model that makes us of similarity.

Overview of simulation model is presented as in Figure 5.1. There may be number of k domains. Moreover, there are several users per domain each of whom

has its own resource broker, and resources. There is a History Management Engine (HME) with scheduling heuristics HISA implemented by each Resource Broker.

The procedures between entities are working as follows:

1. Users send their jobs (Gridlets in GridSim) to their Broker.
2. As soon as simulation is started each resource reports its configurations and availability to all of the GISs (Grid Information Service).
3. The Broker queries and gets available resources with their characteristics from the GIS.
4. After that it gets the Gridlets from the UserEntity (packed in BoT).
5. The Broker assigns tasks to the appropriate resources according to HME decisions. HME decides these mappings according to scheduling heuristic. All tasks are processed by the resources.
6. And then the Broker collects the results from a Resource after execution of an assigned job is complete.
7. When a job is finished, the Broker sends it to its owner, i.e. UserEntity.

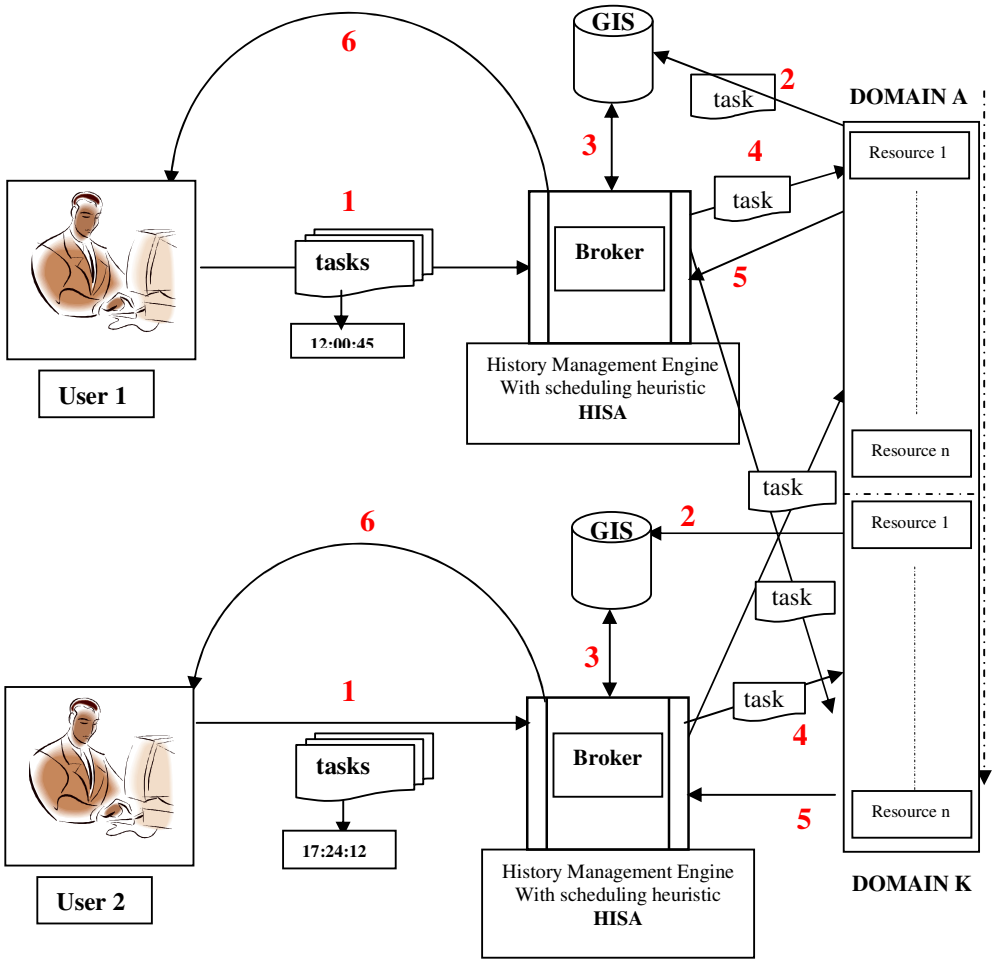

Figure 5.1: MetuGrid Simulation Model adapted from [82]

5.4 HISA (History Injected Scheduling Algorithm)

In this study, to find a similarity between two applications, we created a history file that stores all information generated from the beforehand simulation results. All resources and the list of all Gridlets that assigned to each resource and average value of each significant attribute are stored in the history. And the history is used for computing the owner factor to decide whether it is important or not while defining the value of the similarity. The history is also used for finding the most suitable resource with the help of the similarity value. Newly arrived attributes of Gridlets are compared with those already stored in the history, to find similarity value with the existing ones in the history.

Moreover, history.conf file stores the weight value of Gridlets attributes and initial value of TH (Threshold). These weight parameters are tuned manually by recurring experiments. After each experiment, similarity value is changed according to Gridlets attributes so weight of each attributes is also changed. According to simulation results, while the significances of some attributes may be increased, the others may be decreased.

Before we give the algorithm in Figure 5.2, definition of some expressions and formulas are given as follows:

Computation of the average value of each Gridlets attributes on each resource.

M_i **:** is the number of Gridlets assigned to the resource i, j = 1.....M_i

N **:** is the number of machines, i = 1.....N

K **:** is the number of attributes, k = 1.....K

$g_{i,j}$ **:** j^{th} gridlet previously assigned to resource i.

$g_{i,j}[k]$ **:** k^{th} attribute of gridlet $g_{i,j}$.

C **:** the new gridlet that sholud be assigned to a resource
C[k] **:** k^{th} attribute of gridlet C.

ω_k **:** is a multiplier denoting weight of k^{th} attribute. Manually assigned fixed value.

$Av_i[k] = \sum_{j=1}^{M} g_{i,j}[k]$ is the average of the k^{th} attribute of all Gridlets assigned to resource i

So, the average value of the newly arrived Gridlets attribute is computed by recomputing the average after adding the value of the new Gridlet as:

$$Av_i[k]_{new} = \frac{\left[M \times Av_i[k]_{old}\right] + g_{i,M+1}}{M+1}$$

This operation is used for computation of the average value of each attributes of newly arrived Gridlet in each resource. With the help of this value, we can measure frequency of the Gridlets attributes assigned in a resource and all these average values will be used for finding of the similarity value.

userName is the owner of the newly arrived Gridlet.

ownerName is the owner of the Gridlets that already assigned and executed on the resource in the history.

$$Ow = 1 - \frac{ownerCount}{N}$$

This formula is used to compute the number of Gridlets assigned to a resource by a user until that time. It also compares the owner of newly arrived Gridlets with the owner of Gridlets that already assigned to resources in the history. Hence, **Ow** is the owner value that defines how often the owner of the Gridlet uses this resource. It is computed between the [0, 1]. If the owner factor close to 0, this signifies the owner is highly important when defining the similarity. We know that a user generally works on the same application and makes the same tests on it. Another words, length, inputSize, and outputSize of the user's application can be mostly same and so its runtime can also be the same. For example; a user assigned only one Gridlet to a resource and also the total Gridlets assigned to this resource is just one. Contrary to this, the same user assigned 80 Gridlets to another resource and the total Gridlets assigned to this resource is 100. In the first case the owner value is computed as 0, and in the second case it is computed as 0.2. According to these results, HISA assigns newly arrived Gridlet to the second resource.

ω_k is the Gridlet's attributes weight value acquired from the ***history.conf*** file defines the attribute significance. Weight parameters are tuned manually according to experiments results. When this factor is increased, the significance of the attributes used for definition of the similarity also increases. Contrary to this, if the factor is decreased, the significance of the attribute is also decreased. That is, the values of the attribute significance factor change manually according to the results of the simulations. For example, after a few simulations we assume that the owner factor does not affect the similarity between the two applications. The owner value computed in step 8 in part C. If this value is close to 1 that means the owner value is not important. Hence, we assign zero to the owner factor in the history.conf file, and while defining the similarity value or vice versa.

$$\mathrm{sim}_i = \sum_{k=1}^{N} \left[\left| Av_i[k] - C[k] \right| \times \omega(k) \right]$$

This formula is computed for each resource in the history. It takes absolute value of difference between the value of each attributes of newly arrived Gridlet and the average value of this attribute in the resource and then multiplies this value by the weight of corresponding attribute. This is required to compute the similarity between the newly arrived Gridlet and the Gridlets assigned to the resources in the history. **i** is chosen such that sim_i is maximum.

The HISA Algorithm

(A) Initialization

(1) a = {a_1, a_2, a_3, ….., a_n} is the set of Gridlets' attributes

(2) let TH = Threshold

(3) let H = History Database

(4) Read from H and get initial value of Gridlets attributes

(B) Definition of Similarity

(5) For each attribute compute its average value for its corresponding machine

For (k = 1; k <= n; k++) {where n is the number of Gridlet attributes

$$\mathrm{Av}_i[k]_{\mathrm{new}} = \frac{\left[M \times \mathrm{Av}_i[k]_{old} \right] + g_{i,M+1}}{M+1}$$

(C) Finding of Owner Factor whether it is important or not.

(6) Let ownerCount = 0

(7) Let Ow be ownerFactor

(8) Let current user name be userName and ownerName be Gridlet owner in the H

For (j =1; j<= m; j++) {where m is the number of Gridlets in the H

If (username=ownerName) {

ownerCount++; }

}

$$\mathrm{Ow} = 1 - \frac{ownerCount}{N}$$

r*eturn Ow;*

(D) Finding of Similarity

(9) let sim = currentSimilarity

(10) let initial similarity similarityValue = -1

(11) let initial selectedEntry = null (it stores an information about a history entry whose value is smaller than the value of TH which is the most similar to the current Gridlet)

While (no Resource are left) {

int sim = 0;

Get Next element (resource) from H

currentElement = nextElement;

$$\mathrm{sim}_i = \sum_{k=1}^{N} \left[\left| Av_{\mathrm{i}}[k] - C[k] \right| \times \omega(k) \right]$$

M is the number of the attributes of the Gridlets in the history

(12) If (sim < TH) {

If (selectedEntry==null || sim < similarityValue) {

If there is the current similarity is smaller than the TH value, the resource of the Gridlets which is the most similar to the newly arrived Gridlets is selected. If not, the resource is selected directly.

selectedEntery=currentEntry;

similarityValue=sim;

} }

If (selectedEntry == null) {

return -1 (No suitable resource is found)

*return selectedEntry.get**ResourceID**()*

*(The suitable resource is found and its **ID** is returned for job assignment)*

Figure 5.2: The definition and computation of Similarity in HISA

CHAPTER 6

EXPERIMENTATION AND EVALUATION

6.1 Simulation Model in General in GridSim

GridSim has multi-threaded entities that each of which runs in parallel in its own thread. These entities are used for simulation of different types of processors like single or multiple processors, heterogeneous resources, users, resource brokers (grid schedulers), information service, network based I/O, and statistics [14].

We can summarize these entities and their features as follows:

1. User Entities: A Grid user is represented by a user entity. Each user has some characteristic that separates from the other users in the Grid environment. For example; a user is separated from the other one:

    a. The types of jobs are created like job length, execution time, number of replications etc.
    b. Time zone, that is, different time zones show geographic distribution of users

c. Activity rate shows how often a user creates a new job.
d. Budget and deadline factor. They are measured in the range [0,1]. For example if a deadline factor close to 0, user wants the slowest resources because of low priority deadline, on the other hand if a budget factor is close to 1, user wants the most expensive and fastest resources because he/she is willing to spend as much money as possible.

2. Resource Entities: A Grid resource is represented by an instance of the Resource entity. Each resource entity has some characteristics:
    a. Each resource may have different number of processors.
    b. Processors' speed and cost can be different on each resource. The processor or resource speed can be defined in terms of MIPS, SPEC, and the standard machine.
    c. Different local load factor can be assigned on resources.
    d. Each resource can be modeled with time-shared or space-shared systems.
    e. Each resource can be located in a different time zone.
3. Resource Broker Entities: It is also called grid scheduler. Each user first submits its job to its broker that is connected to the user. Then the broker gets a list of available resources from the grid information service. After that the task is scheduled to the most suitable resource by the broker according to the user's scheduling policy. There is always a competition while the brokers access the resources. Because the aim of each broker is to optimize the policy of its users.
4. Grid information Service: All resources in the grid environment register themselves to the Grid Information Service. It also keeps track of a list of available resources. This list can be queried by the brokers to gain contacts, configurations, and status information of the available resources [14].

These GridSim entities communicate using events. Events are used for

service request and delivery. There are two type of events; internal and external event. Internal events are created by the same entity. Contrary to this, external events are originated from external entities. Internal events must be asynchronous to avoid the deadlock. On the other hand, an external event can be both synchronous and asynchronous. An asynchronous event is an event created by the entity, such as the entity is continues with its other activities without waiting the completion of the asynchronous event. However, a synchronous must waited for until its completion.

To create and simulate a Grid environment and analyze scheduling algorithms, we should do following steps:

1- Firstly, Grid resources are created with different configurations and capabilities. And then users are also created with different requirements such as QoS requirements.
2- Secondly, a number of Gridlets are created for model applications with all the associated parameters.
3- Thirdly, a GridSim user entity is defined to create and interact with the resource broker scheduling entity which is used to coordinate execution. To submit or receive processed Gridlets and acquire Grid information, it can directly interact with GIS and resource entities.
4- Finally, a resource broker entity is implemented to perform application scheduling on Grid resources.

In order to simulate a Grid environment, a general simulation model is created as seen from Figure 6.1. In a general simulation model, each user has its own resource broker with I/O queue responsible for assignment of jobs and optimization of their requirements and objectives. The resource broker entity queries GIS entity for discovery of the available resources and selects suitable resources according to user's request. The GIS entity returns a list of registered resources, and their contact details. And then resources send dynamic information

such as resources cost, capability, availability, current load, and other configuration parameters to the resource broker. The resource broker gets all jobs from the users and puts them in a job queue according to FCFS. All these jobs in the queue are scheduled according to scheduling policy embedded in the resource broker. And then the appropriate resources are selected by the resource broker, and user jobs (Gridlets) are sent to the I/O queue of the resources for execution. The resources send back the processed Gridlets to the I/O queue of the broker entity. Finally, the processed Gridlets are collected from the I/O queue by the user.

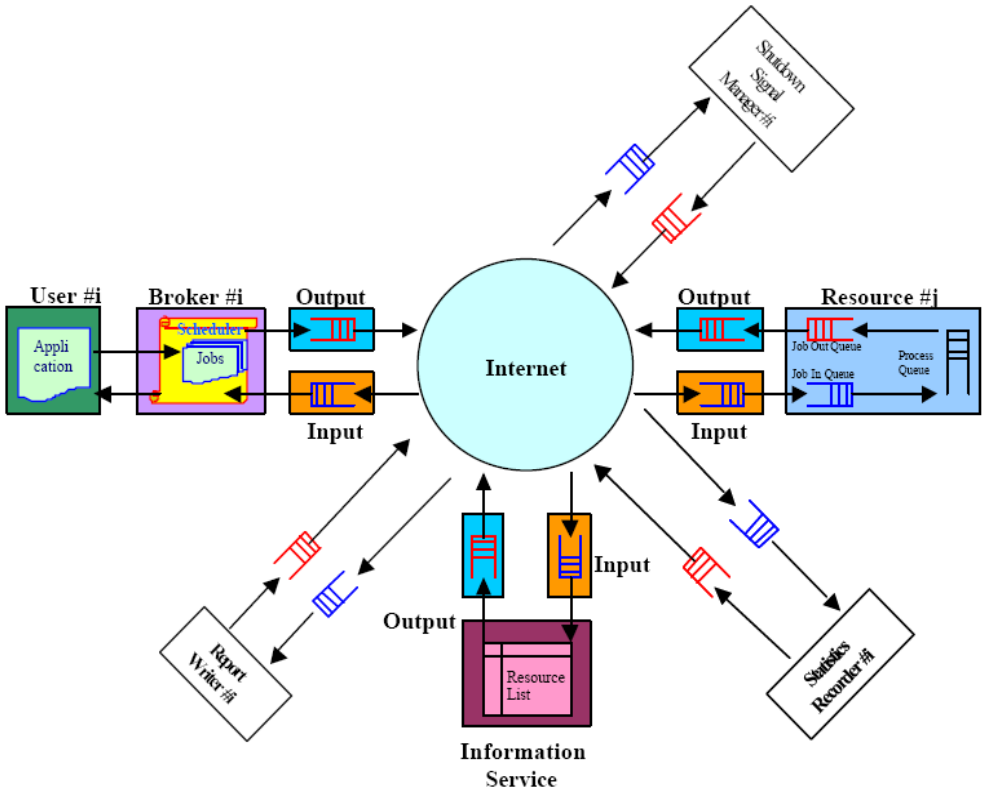

Figure 6.1: A general simulation model in GridSim with its entities [14].

6.2 HISA Simulation Environment

When we develop HISA, we created a simulation Grid environment like in the general simulation model in GridSim given in Figure 6.1.

We craeted data from 4 users to 100 users with different Gridlets proporties, length, size and different scheduling policies. As we mentioned eralier, in GridSim the properties of a task is repsresented by a Gridlet object as seen from Table 6.1.

Table 6.1: A Sample Gridlet Object.

Gridlet ID	Gridlet Lengt (MI)	Input File Size (byte)	Output File Size (byte)
1	3955	58	5
.....	.....	.....	.....
N	.....	.....	.....

These Gridlets are owned by a user with different requirements. That means while we created a user, we also defined its number of Gridlets, connection speed (baud rate), maximum time to run simulation (actually it is the deadline when the simulation will end no matter what is the Gridlet execution status), and scheduling policy like our HISA, Time, Cost, and Cost-Time minimization as seen from Table 6.2.

Table 6.2: A Sample User Object.

User Name	Baud Rate (Mbit/s)	Max. Simulation Time (hour)	Scheduling Policy	# of Gridlets
User_1	213	13 hour	HISA	63
.....	.....	.....	.....	.....
User_N	.....	.....	.....	.....

In this work, we used GridSim default broker as a resource broker. We create a history management engine method responsible for history and embeded it in scheduler adviser method in GridSim broker class. HISA finds the most suitable resource based on the similarity value and sends it to the history management engine. And then it assigns the Gridlet to the found resource.

Before the simulation start, a user creates an experiment that acts as a placeholder. It is composed of GridletList that stores a set of Gridlets to be processed and user requirements with a scheduling policy. Whenever simulation start the broker creates a resource list to store dynamic information and characteristic properties of available resources acquired from the GIS. During the simulation each broker continuously queries the GIS and gets dynamic information about the available resources and load on these resources. And then user sends its application to its resource broker via the application interface. The broker of each user gets their Gridlets from its experiment object via experiment interface of that broker. After that the broker of user puts all Gridlets to be sent for execution into the unfinished GridletList. In our design, there is no time management; that means each user wait specific delay and then sends all Gridlets at one moment packed in experiment object to its brokers. This delay is defined when the users are created. Since we developed a static scheduling algorithm, there will not be a new Gridlet during the simulation. That means new users can not send new Gridlets until this

simulation finishes. Then the broker starts scheduling and schedules all of Gridlets one by one. The Gridlets in each broker are scheduled according to the requested scheduling heuristic, status of resources and their availability. Our history management engine selects the most suitable resource for the Gridlets according to similarity value computed by the HISA. And then Gridlets that are mapped to a specific resource are added to the GridletList by the broker. A distpatcher embedded in each broker selects the number of Gridlets from the GridletList and assigns them to the reosurce according to that resource is not overloaded. And also it decides how many Gridlets to be send based on the type of resources. Because we created different type of resources in terms of time and space shared, all Gridlets executed depending on the resource type that assigned to them. For example, if a new job arrive a space-shared system, the job is executed if and only if there is a free PE available in the resource. If there is no available PE for the newly arrived job, it is sent resource queue. On the other hand, a job arrives at a time-share system, the execution of jobs is started immediately by the time-shared system. All resources in the systems are shared between the jobs. Hence a new Gridlet assigned to a resource according to its current load. When a new Gridlet arrives, the processing time of existing Gridlets is updated. After that the newly arrived job is added to the execution set. The resource sends back the completed Gridlets to the receptor of its broker created when the simulation start. And then updated experiment data with processed Gridlets are returned back to the user entity by the broker.

6.3 Comparison of HISA with Three Scheduling Algorithms Embedded in GridSim

In order to evaluate and analysis our scheduling algorithm HISA, we compared it with there scheduling algorithms embedded in GridSim simulation tool. These are Time Minimization, Cost Minimization, and Cost-Time Minimization.

6.3.1 Time Minimization

The aim of the Time Minimisation scheduling algorithm is to complete the task as quickly as possible within the budget available [16]. This algorithm has four steps:

1- For an assigned job, the next completion time is calculated taking the previously assigned job into account. This procedure is done for each resource.
2- All resources are sorted according to the next completion time.
3- A job is assigned to the first resource in the list in which the remaining budget per job is greater than the cost per job.
4- The steps are repeated until all jobs are finished.

6.3.2 Cost Minimization

In the Cost Minimization scheduling algorithm, the job is completed as economically as possible within the given deadline [16]. This algorithm has two steps:

1- All resources are sorted according to the cost in increasing order.
2- As many as possible jobs are assigned to the resource within the deadline. This is done for each resource in the order.

6.3.3 Cost-Time Minimization

The purpose of the Cost-Time Minimization scheduling algorithm is optimizing the time without incurring additional processing expenses [17].

This algorithm has nine steps:

1- All resources are sorted by increasing order of cost. If two or more resources have the same cost, order them by increasing order of MIPS.
2- The resources with the same cost are grouped.
3- The resources in the group are sorted by increasing order of cost.
4- If any jobs that are assigned to the resources but not dispatched to the resource for execution, a number of jobs are moved to the Unassigned-Job-List. This is used in updating the whole schedule based on latest resource availability information.
5- A job is selected from the Unassigned-Job-List.
6- For an assigned job, the next completion time is calculated/predicted taking the previously assigned job into account. This procedure is done for each resource.
7- All resources are sorted according to the next completion time.
8- If the predicted job completion time is less than the deadline, the job is assigned to the first resource and then removed from the Unassigned-Job-List.
9- Repeat from step 4 to step 8 for each job in the Unassigned-Job-List depending on the processing cost and the budget availability.

Our evaluation and analysis consist of three different parts:

1- The first part is comparison of Gridlets completion time on each reosurce. In this simulation, we performed nearly 800 simulations.
2- The second part is comparing of total number of Gridlets assigned succesfuly. In this part, we also performed nearly 800 simulations.
3- The last part is computation of simulation duration time for each algorithm, analysis of the results and then comparing of the algorithms in terms of simulation duration time. Actually the simulation duration time is the wall clock time of these algorithms. To do this, we performed nearly 400 simulations.

The first part is comparing of Gridlets completion time. In this simulation, we performed nearly 800 simulations. Moreover, we used 9 resources with different characteristics. Each of them has different MIPS rating, CPU models, number of PEs, and operating systems. MIPS rating of all PE are acquired from the SPEC CPU (INT) 2000. These resources are used for all simulations. The aim of this simulation is to compare the completion time of Gridlets on the resources between HISA and other three algorithms.

In this simulation, we performed different values of Gridlets for a single user. The Gridlet's length is varied from 100 MI to 10000 MI. Gridlets' length and their completion time obtained from the experiment are plotted in Figure 6.2.

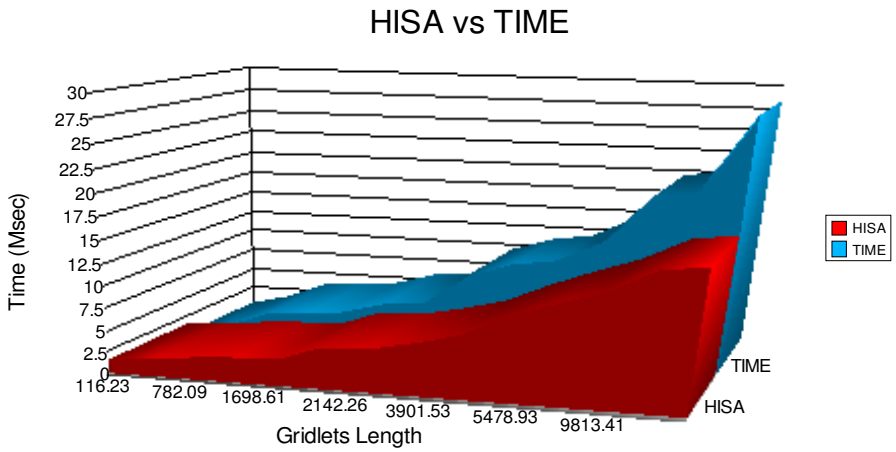

Figure 6.2: Comparison of Gridlets completion time: HISA and Time Minimization

It can be observed that at the begining of the simulation, both algorithms showed nearly same performance. However, when the number of experiments is increased, we observed that HISA pointed out better performance. Because HISA is

based on history aproach, after each experiment the history file gets rich. It means that, after each simulation HISA writes beforehand experiments' results in this history file. Before the algorithm assigns the Gridlets in the subsequent simulation, it makes a deep analysis through this file to find the most suitable resource by using the similarity value. It is more likely that the algorithm finds the most suitable resource assigned to Gridlet by using such an advanced history file. The resource is found executing the Gridlets fastest. Therefore, Gridlets assigned by the HISA to be completed as fast as possible. The Figure 6.2 shows that the application scheduling using HISA is more successful than the others in terms of Gridlets completion time. The data on which Figure 6.2 is based is given in Table 6.3. For example; as seen from the Table 6.3 while the Gridlet with 9813.41 MI length is completed by the HISA in 15.32 ms, it is completed in 20.27 ms by the Time Minimization scheduling algorithm. Moreover as seen from Figure 6.2, HISA is more successful than the others when the Gridlet size increases.

Table 6.3 shows the Gridlets completion time for both algorithms. The column 1 depicts the simulation order and the column 2 depicts length of Gridlets in terms of MI (Million of Instructions) each of which has the same characteristic. The others columns show completion time of Gridlet on the reosurces for each algorithm in terms of millisecond.

Moreover, we can see such favourable results between HISA and Cost Minimization, and Cost-Time Minimization algorithms, from Figure 6.3, Figure 6.4 and Table 6.4, Table 6.5 respectively.

Table 6.3: Comparison data on the Gridlets completion times: HISA and Time Minimization.

Sim. Order	Gridlet length (MI)	HISA (msec)	TIME (msec)
1	116.23	1.52	0.38
2	510.21	2.01	1.45
3	782.09	2.51	3.34
4	882.78	2.88	3.77
5	1698.61	3.04	4.41
6	2124.29	4.89	5.56
7	2142.26	4.99	6.12
8	2512.66	5.98	9.57
9	3901.53	7.12	11.14
10	4528.58	9.38	11.85
11	5478.93	11.31	14.33
12	6470.97	12.78	19.49
13	9813.41	15.32	20.27
14	9937.33	16.01	28.39

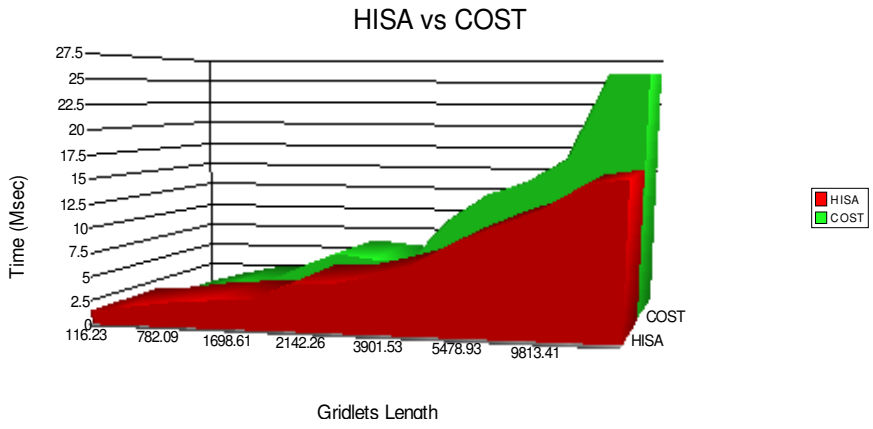

Figure 6.3: Comparison of Gridlets completion time: HISA and Cost Minimization

Table 6.4: Comparison data on the Gridlets completion times: HISA and Cost Minimization.

Sim. Order	Gridlet Length (MI)	HISA (msec)	COST (msec)
1	116.23	1.52	0.31
2	510.21	2.01	1.33
3	782.09	2.51	2.23
4	882.78	2.88	2.52
5	1698.61	3.04	4.41
6	2124.29	4.89	6.06
7	2142.26	4.99	5.61
8	2512.66	5.98	5.28
9	3901.53	7.12	10.21
10	4528.58	9.38	12.93
11	5478.93	11.31	14.33
12	6470.97	12.78	16.93
13	9813.41	15.32	25.68
14	9937.33	16.01	26.01

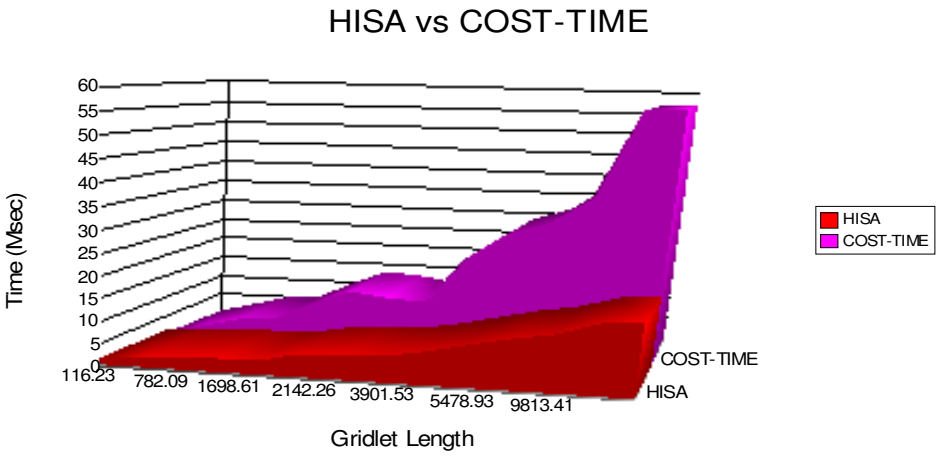

Figure 6.4: Comparison of Gridlets completion time: HISA and Cost-Time Minimization.

Table 6.5: Comparison data on the Gridlets completion times: HISA and Cost-Time Minimization.

Sim. Order	Gridlet Length (MI)	HISA (msec)	COST-TIME (msec)
1	116.23	1.52	0.68
2	510.21	2.01	2.93
3	782.09	2.51	4.91
4	882.78	2.88	5.54
5	1698.61	3.04	9.7
6	2124.29	4.89	13.33
7	2142.26	4.99	12.32
8	2512.66	5.98	11.62
9	3901.53	7.12	22.46
10	4528.58	9.38	28.45
11	5478.93	11.31	31.53
12	6470.97	12.78	37.25
13	9813.41	15.32	56.5
14	9937.33	16.01	57.22

In the second part of simulation, we compared these four algorithms in terms of a number of Gridlets assigned successfuly. For each algorithm, we created data from 4 users to 100 users with same Gridlets proporties and different scheduling policies. The aim of this simulation is to test the achievement of algorithms in terms of the number of Gridlets assigned successfuly. In the beginning of this simulation, firstly the number of Gridlets created for each user and then all are summed for each simulation. After each simulation, according to simulation results, we computed the number of Gridlets assigned successfuly and. When we averaged all test results for each algorithm and each simulation, we observed that HISA gave the second best result among the others in terms of the humber of Gridlets assgined successfuly. While the Cost Minimization algorithm

was the best one, the Time Minimization algorithm was the worst. On the other hand, Cost and Time algoritm was close to our algorithm as seen from Figure 6.5, Table 6.6 and Table 6.7.

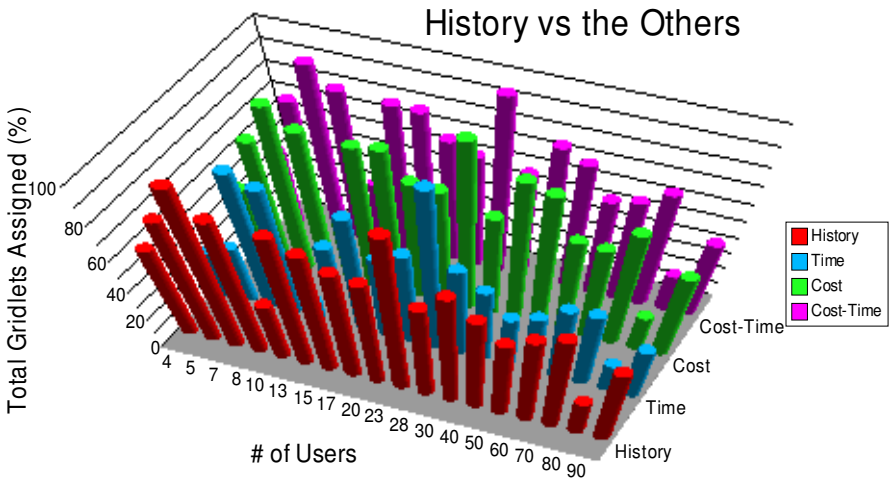

Figure 6.5: Comparison of number of Gridlets assigned successfuly: HISA, Time, Cost, and Cost and Time.

In Table 6.6, the first column shows the simulation order, the second column shows the number of users used in each simulation and the third column shows the number of Gridlets produced by each user. The other columns show the number of Gridlets processed successfuly by each algorithm.

Table 6.6: The number of Gridlets assigned successfuly by each algorithms.

Sim. Order	# of Users	Total Gridlets	History	Time	Cost	Cost-Time
1	**4**	**173**	100	52	77	77
2	**5**	**244**	190	88	181	181
3	**7**	**323**	319	269	304	304
4	**8**	**416**	354	324	354	353
5	**10**	**456**	174	52	173	173
6	**13**	**768**	645	401	642	642
7	**15**	**501**	390	366	428	416
8	**17**	**767**	549	404	550	550
9	**20**	**979**	679	579	692	653
10	**23**	**1192**	1192	1176	1184	1183
11	**28**	**1405**	887	811	884	891
12	**30**	**1399**	1042	680	1202	1132
13	**40**	**1952**	1257	673	1605	1464
14	**50**	**2113**	1132	833	1275	1244
15	**60**	**2845**	1733	1411	1686	1754
16	**70**	**4067**	2707	2045	2935	2829
17	**80**	**3746**	967	727	869	877
18	**90**	**5008**	2690	1765	2717	2390

On the other hand, in the Table 6.7, the same simulation results, this time, are given as the success percentage for each algorithm

In Table 6.7, the first column shows the simulation order, the second column shows the number of users used in each simulation and the third column shows the number of Gridlets produced by each user as the same in Table 6.6. The other columns show the number of Gridlets assigned successfuly by each algorithms in terms of the success percentage for each algorithm.

Table 6.7: The number of Gridlets assigned successfuly by each algorithms in terms of the success percentage for each algorithm.

Sim. Order	# of Users	HISA (%)	TIME (%)	COST (%)	COST-TIME (%)
1	**4**	57.8 %	30.06 %	44.51 %	44.51 %
2	**5**	77.87 %	36.07 %	74.18 %	74.18 %
3	**7**	98.76 %	83.28 %	94.12 %	94.12 %
4	**8**	85.1 %	77.88 %	85.1 %	84.86 %
5	**10**	38.16 %	11.4 %	37.94 %	37.94 %
6	**13**	83.98 %	52.21 %	83.59 %	83.59 %
7	**15**	77.84 %	73.05 %	85.43 %	83.03 %
8	**17**	71.58 %	52.67 %	71.71 %	71.71 %
9	**20**	69.36 %	59.14 %	70.68 %	66.7 %
10	**23**	100 %	98.66 %	99.33 %	99.24 %
11	**28**	63.13 %	57.72 %	62.92 %	63.42 %
12	**30**	74.48 %	48.61 %	85.92 %	80.91 %
13	**40**	64.4 %	34.48 %	82.22 %	75 %
14	**50**	53.57 %	39.42 %	60.34 %	58.87 %
15	**60**	60.91 %	49.6 %	59.26 %	61.65 %
16	**70**	66.56 %	50.28 %	72.17 %	69.56 %
17	**80**	25.81 %	19.41 %	23.2 %	23.41 %
18	**90**	53.71 %	35.24 %	54.25 %	47.72 %
	AVERAGE %	**67.95 %**	**50.51 %**	**69.27 %**	**67.8 %**

In the last part of simulation, for each algorithm, we craeted the same data, from 4 users to 100 users with different Gridlets properties and scheduling policies. Each of user has its own characteristics. For example each user has different Gridlet length, input, and output size etc.

The aim of this simulation is to measure the simulation duration time of each algorithms in terms of the wallclock time.

Figure 6.6 depicts comparison of simulation duration time between HISA and the other three scheduling algorithms. As seen in Figure 6.6, HISA shows poorer performance than Time minimisation. Because HISA is based on history approach, it makes a deep search through a file to find the most suitable resource with the help of similarity value. Whenever simulation starts, HISA creates a history file that stores all information about simulation results. The results produced after each simulation are stored at the end of that history file. Therefore the history file gets bigger after each simulation. Because the search and comparisons take a long time in such a big file, simulation time of HISA is expected to be longer which is confirmed by these experiments.

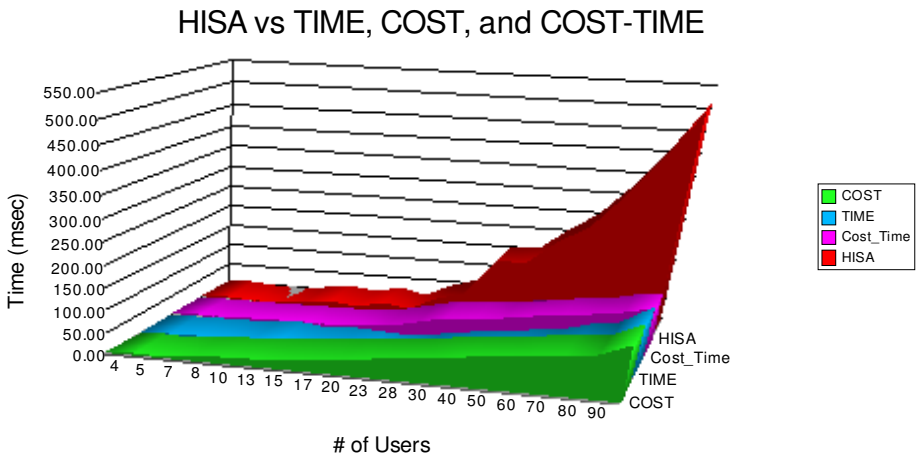

Figure 6.6: Comparison of simulation duration times in terms of wallclock time: HISA, Time, Cost, and Cost-Time minimization.

The data on which Figure 6.6 is based is given in Table 6.8.

In the Table 6.8, the first column shows the simulation order, the second column shows the number of users and the others columns show the simulation completion time in terms of milliseconds for each algorithm. Each user has different number of Gridlets.

Table 6.8: Comparison data on the simulation duration times in terms of wallclock time: HISA, Time, Cost, and Cost-Time minimization.

Sim. Order	# of Users	COST (msec)	TIME (msec)	COST-TIME (msec)	HISA (msec)
1	**4**	3.60	3.40	3.75	4.14
2	**5**	6.70	5.60	3.59	6.50
3	**7**	6.20	6.80	3.91	2.50
4	**8**	10.10	4.75	5.67	0.98
5	**10**	15.70	7.79	7.66	8.76
6	**13**	19.20	12.58	7.50	21.17
7	**15**	23.40	3.60	7.81	9.40
8	**17**	29.80	9.62	10.94	26.83
9	**20**	35.90	7.62	17.56	13.90
10	**23**	43.90	2.40	34.21	44.88
11	**28**	51.60	13.56	49.25	66.18
12	**30**	60.10	26.68	51.16	153.09
13	**40**	67.90	32.45	60.12	152.01
14	**50**	74.50	41.92	71.23	205.92
15	**60**	82.90	50.41	77.29	250.68
16	**70**	90.02	69.54	87.16	324.56
17	**80**	98.34	78.97	99.18	412.21
18	**90**	127.56	123.25	112.51	512.58

CHAPTER 7

CONCLUSION

In this study, a history based scheduling algorithm was designed and implemented for heterogeneous Grid computing systems. The proposed algorithm primarily makes use of history based runtime estimation. The history stores information about the previous executions of the applications. The aim is to define and find similarity between the new application execution requests and the previous application executions to assign the job to the most suitable resource.

The effectiveness of our scheduling heuristic was studied through comparison of its performance with that of the other scheduling algorithms embedded in GridSim. A number of intensive experiments with various test configurations have been conducted. The simulation results presented clearly show that our scheduling algorithm HISA has better performance than the other three algorithms in terms of Gridlets completion time. Moreover, we show that the proposed algorithm gives better results than the others, especially when the history file gets rich in terms of number of re-executions. On the other hand, it gives poorer results than the others in terms of simulation duration time. This occurs when the history file gets bigger and search in a big file takes longer time. We also observed

that our history based scheduling algorithm gave the second best result among the others in terms of number of Gridlets assigned successfully.

As the future work, a more elaborate runtime estimation algorithm may be developed making use of HISA. Moreover, the simulation duration time of HISA may be decreased using efficient data structure and search algorithms. For example grouping the history entries according to their similarities and when one entry does not match, skipping the similar history entries without comparing with the Gridlet to be scheduled would cut down the simulation time. In this case however, an advanced calculation needs to be done to guarantee the mismatch. Hashing resources in the history file can also greatly improve the performance regarding the simulation time.

REFERENCES

[1] Abraham, R. Buyya, and B. Nath, "Nature's Heuristics for Scheduling Jobs on Computational Grids", Proceedings of 8th IEEE International Conference on Advanced Computing and Communications, 2000.

[2] Aggarwal and R. D. Kent, "An Adaptive Generalized Scheduler for Grid Applications", PCS '05: Proceedings of the 19th International Symposium on High Performance Computing Systems and Applications, 2005.

[3] Aggarwal, R. D. Kent, and A. Ngom, "Genetic Algorithm Based Scheduler for Computational Grids", HPCS'05: Proceedings of the 19th International Symposium on High Performance Computing Systems and Applications, 2005.

[4] Ali, A. Anjum, J. Bunn, R. Cavanaugh, F. Lingen, R. McClatchey, M. A. Mehmood, H. Newman, C. Steenberg, M. Thomas and I. Willers, "Predicting the Resource Requirements of a Job Submission", Computing in High Energy Physics, 2004.

[5] Amin, G von Laszewski and A.R. Mikler, "Grid Computing for the Masses: An Overview," in Grid and Cooperative Computing (GCC2003), China, pp. 464-473, December 2003.

[6] Anderson, J. Cobb and E. Korpela, "SETI@home: An Experiment in Public-Resource Computing", Communication of the ACM, vol. 45, no. 11, pp 56-61, November 2002.

[7] Armstrong, D Hensgen, and T. Kidd, “The Relative Performance of Various Mapping Algorithms is Independent of Sizable Variances in Run-time Predictions, 7th IEEE Heterogeneous Computing Workshop (HCW '98), pp. 79-87, Mar. 1998.

[8] Arnold, S. Agrawal, S. Blackford, J. Dongarra, M. Miller, K. Seymour, K. Sagi, Z. Shi, and S. Vadhiyar, “User Guide the NetSolve”, Innovative Computing Dept. Technical Report, University of Tennessee, 2002.

[9] Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, B. Park, and H. Song, “Parsec: A Parallel Simulation Environment for Complex Systems”, IEEE Computer, Oct 1998.

[10] Baraglia, R. Ferrini, and P. Ritrovato, “A Static Mapping Heuristic for Mapping Parallel Applications to Heterogeneous Computing Systems”, Technical Report, March 2003.

[11] Berstis, “Fundamentals of Grid Computing”, IBM RedBooks Paper, November 2002.

[12] Braun, H. J. Siegel, and N. Beck, “A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems”, Journal of Parallel and Distributed Computing, 2001.

[13] Braun, H. J. Siegely, N. Becky, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, “A Comparison Study of Static Mapping Heuristics for a Class of Meta-tasks on Heterogeneous Computing Systems”, Heterogeneous Computing Workshop, 1999 (HCW '99) Proceedings Eighth, 12 April pp. 15 – 29, 1999.

[14] Buyya and M. Murshed, “GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing”, Concurrency and Computation: Practice and Experience, 2002.

[15] Buyya, D. Abramson, and J. Giddy, “Nimrod-G: An architecture for a resource management and scheduling system in a global computational grid”, In Proc. of the 4th Intl. Conference & Exhibition on high Performance Computing in Asia-Pacific Region (HPC Asia’00), Beijing, China, May 2000.

[16] Buyya, J. Giddy, and D. Abramson, “An Evaluation of Economy-based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications”, Proceedings of the 2nd International Workshop on Active Middleware Services (AMS 2000), Kluwer Academic Press, August 1, 2000.

[17] Buyya, M. Murshed, D. Abramson, “A Deadline and Budget Constrained Cost-Time Optimization Algorithm for Scheduling Task Farming Applications on Global Grids”, GridSim Toolkit Release Document, December 2001.

[18] Cameron, R. Carvajal-Schiaffino, A. P. Millar, C. Nicholson, K. Stockinger, and F. Zini, “UK Grid Simulation with OptorSim”, 2001.

[19] Caniou and E. Jeannot, “Experimental Study of Multi-Criteria Scheduling Heuristics for GridRPC Systems”, In ACM/IFIP/IEEE Euro-Par-2004: International Conference on Parallel Processing, LNCS 3149, Pisa, Italy, pp. 1048-1055, August 31 - September 3 2004.

[20] Cao, D. Spooner, and G. Nudd, “Agent-based Resource Management for Grid Computing”, Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID.02), IEEE 2002.

[21] Casanova, A. Legrand, D. Zagorodnov, and F. Berman, “Heuristic for Scheduling Parameter-Sweep Applications in Grid Environments”, IEEE, 2000.

[22] Casanova, “SimGrid: A Toolkit for the Simulation of Application Scheduling”, Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001), Brisbane, Australia, May 15-18, 2001.

[23] 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.

[24] Chan, “The Network Simulator - ns-2”, http://www.isi.edu/nsnam/ns, January 03, 2006.

[25] Chang and M. Livny, “Distributed Scheduling under Deadline Constraints: A Comparison of Sender-Initiated and Receiver-Initiated Approaches”, IEEE Real-Time Systems Symposium, 1986.

[26] Chen, N. S. Flann, and D.W. Watson, “Parallel Genetic Simulated Annealing: A Massively Parallel SIMD Approach”, IEEE Trans. Parallel Distrib. Comput. 9, 126-136, 2 Feb 1998.

[27] Chtepen, B. Dhoedt, and P. Vanrolleghem, “Dynamic Scheduling in Grid Systems”, 6th FTW PHD Symposium, Interactive Poster Session, Gent, Belgium, November, 2005.

[28] Cirne, D. Paranhos, L. Costa, E. S. Neto, N. Abílio, N. Andrade, F. Brasileiro, and J. Sauvé, “Using MyGrid to Run Bag of Tasks Applications on Computational Grids”, In International Conference on Parallel and Distributed Computing (Euro-Par), Lecture Notes in computer Science, volume 2790, pp 169-180, 2003.

[29] Cirne, D. Paranhos, L. Costa, E. Santos-Neto, F. Brasileiro, J. Sauvé, F. Silva, C. Osthoff, and C. Silveira, “Running Bag-of-Tasks Applications on Computational Grids: The MyGrid Approach”, Proceedings of the 32nd International Conference on Parallel Processing (ICPP 2003), 6-9 October 2003, Kaohsiung, Taiwan, IEEE Computer Society, 2003.

[30] Claassens and E. van der Weegen, “Grid Computing, Lecture Notes of Development of Large Software Systems”, p.p: 7-8 Radboud University.

[31] Condor Project Homepage, “Condor: High Throughput Computing”, http://www.cs.wisc.edu/condor/description.html, Last date accessed: October 26, 2006.

[32] Costa, L. Feitosa, E. Araujo, G. Mendes, R. Coelho, W. Cirne, and D. Fireman. “MyGrid: A complete solution for running bag-of-tasks applications”, In Proc. of the SBRC 2004 – Salao de Ferramentas 22nd Brazilian Symposium on Computer Networks – III Special Tools Session, May 2004.

[33] Crosby, http://www.hep.ucl.ac.uk/~pac/EDGSim/edgsim.html, Last date accesses: January 29, 2003.

[34] Crosby, "Measurement and Simulation of the Performance of high Energy Physics Data Grids", Submitted to the University of London in fulfillment of the requirements for the award of the degree of Doctor of Philosophy, page(s):79-82, 2004.

[35] Dong and S. G. Akl, "Scheduling Algorithms for Grid Computing: State of the Art and Open Problems", Technical Report No: 2006/504, Jan 2006.

[36] European Union, The DataGrid Project, http://www.cern.ch/eu-datagrid, Last date accessed: March 31, 2005.

[37] Ferrira, V. Bersis, J. Armstrong, M. Kendzlerski, Andreas Neukoetter, M. Takagl, R. Bing-Wo, A. Amir, R. Murakawa, O. Hernandez, J. Magowan, and N. Bleberstein, "Introduction the Grid Computing with Globus", IBM RedBooks Paper, International Technical Support Organization , September 2003.

[38] Foster, C. Kesselman, and S. Tuecke, "The Anotomy of the Grid", Intl J. Supercomputer Applications, 2001.

[39] Freund, M. Gherrity, S. Ambrosius, et al, "Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet", pp.184–199, In Proc., The 7th IEEE Heterogeneous Computing Workshop (HCW '98), Orlando, Florida, USA, 1998.

[40] Freund, T. Kidd, and L. Moore, "SmartNet: A Scheduling Framework for Heterogeneous Computing", Proc. 2nd Int. Symp. Parallel Architectures, Algorithms, and Networks, (1996), pp. 514-521, 1996.

[41] Fuegi, LSF Basics, http://scv.bu.edu/Graphics/lsf.html, Last date accessed: June 06, 2006.

[42] Fujimato and K. Hagirhara, "Near-Optimal Dynamic Task Scheduling of Independent Coarse-Grained Tasks onto a Computational Grid", In 32nd Annual International Conference on Parallel Processing (ICPP-03), pp.391-398, China, 2003.

[43] Greenshields, “Grid Computing in a Biomedical Context, Clinical Center”, Clinical Pathology Conference, Mart 21, 2002.

[44] Grid.Org, Grid Computing: The Basics http://www.grid.org/about/gc, Last date accessed: November 09, 2006.

[45] Gu, S. Zhang, and S. Li, “Grid Workflow based on Dynamic Modeling and Scheduling”, ITCC '04: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04) Volume 2, IEEE, 2004.

[46] Huedo, R. S. Montero, and I. M. Llorente, “Experiences on Adaptive Grid Scheduling of Parameter Sweep Applications”, pdp, p. 28, 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP'04), 2004.

[47] Iamnitchi, M. Ripeanu, and Ian Foster, “Locating Data in (Small-World) Peer-to-Peer Scientific Collaborations”, Electronic Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02), March 7-8, 2002.

[48] Ibarra and C. E. Kim, “Heuristic Algorithms for Scheduling Independent Tasks on Non-Identical Processors”, J. Assoc. Comput. March 24, 1977.

[49] Iverson, F. Özgüner, and G. J. Follen, “Parallelizing Existing Applications in a Distributed Heterogeneous Environment”, In Proc. IPSS Workshop on Heterogeneous Computing, Santa Barbara, CA, IEEE, Computer Society, April, 1995.

[50] Jacob, L. Ferreira, N. Bieberstein, C. Gilzean, J. Girard, R. Strachowski, S. Yu, “Enabling Applications for Grid Computing with Globus”, IBM Redbook, IBM Corp. June 2003.

[51] Jin, J. Liu, and Z. Yang, “Modeling Agent-Based Task Handling in A Peer-to-Peer Grid”, Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT’04), 2004.

[52] Kim and J. B. Weissman, "A Genetic Algorithm based Approach for Scheduling Decomposable Data Grid Applications", Proceedings of the 2004 International Conference on Parallel Processing (ICPP'04), 2004.

[53] Kim, J. S. Kim, M. Li et al., "An Online Scheduling Algorithm for Grid Computing Systems", GCC 2003, LNCS 3033, pp. 34–39, Springer-Verlag Berlin Heidelberg, 2004.

[54] Kim, K. Kim, S, Shivle, H. J. Siegel, A. A. Maciejewski, T. D. Braun, M. Schneider, S. Tideman, R. Chitta, R. B. Dilmaghani, R. Joshi, A. Kaul, A. Sharma, S. Sripada, P. Vangari, and S. S. Yellampalli, "Dynamic Mapping in a Heterogeneous Environment with Tasks Having Priorities and Multiple Deadlines", IEEE, 2003.

[55] Kolodner, Case-Based Reasoning, Morgan Kaufmann Publishers, 1993.

[56] Komorowski, L. Polkowski, and A. Skowron. Rough sets: A Tutorial. In S.K. Pal and A. Skowron, editors, Rough-Fuzzy Hybridization: A New Method for Decision Making, Springer-Verlag, Singapore (in Print), 1998.

[57] Krishnaswamy, S. W. Loke, and A. Zaslavsky, "Estimating Computation Times of Data-Intensive Applications", IEEE Distributed Systems Online, 2004 Published by the IEEE Computer Society Vol. 5, No. 4; April 2004.

[58] Laguna and F. Glover, Tabu Search, http://www.tabusearch.net/, Last date accessed: April 15, 2004.

[59] Liu, "Grid Scheduling", Department of Computer Science University of Iowa.

[60] Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic Matching and Scheduling of Class of Independent Tasks onto Heterogeneous Computing Systems", 8th IEEE HCW, April 1999.

[61] Maheswaran, T. D. Braun, and H. J. Siegel, "Heterogeneous Distributed Computing", Encyclopedia of Electrical and Electronics Engineering, J. G. Webster, editor, John Wiley & Sons, New York, NY, 1999 Vol. 8, pp. 679-690, 1999.

[62] Menasce, D. Saha, S. C. da Silva Porto, V. A. F. Almeida, and S. K. Tripathi, "Static and Dynamic Processor Scheduling Disciplines in Heterogeneous

Parallel Architecture", Journal of Parallel and Distributed Computing, vol. 28, 1995.

[63] Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller, "Equation of State Calculations by Fast Computing Machines", J. Chem. Phys.,21, 6, 1087-1092, 1953.

[64] Moreno, "Job Scheduling and Resource Management Techniques in Dynamic Grid Environments", 1st European Across Grid Conference, Madrid, February, 2003.

[65] Muthuvelu, J. Liu, and N. L. Soe, "Task Scheduling for Coarse-Grained Grid Application", Grid Computing and Distributed Systems Laboratory Department of Computer Science and Software Engineering The University of Melbourne, Australia. 2005.

[66] Muthuvelu, J. Liu, N. L. Soe, S. Venugopal, A. Sulistio and R. Buyya, "A Dynamic Job Grouping-Based Scheduling for Deploying Applications with Fine-Grained Tasks on Global Grids", Grid Computing and Distributed Systems (GRIDS) Laboratory Department of Computer Science and Software Engineering The University of Melbourne, Australia, 2005.

[67] NetSolve, NetSolve/GridSolve, http://icl.cs.utk.edu/netsolve, Last date accessed: September 2006.

[68] Pawlak, Rough Sets: "Theoretical Aspects of Reasoning about Data", Kluwer Academic Publishers, 1992.

[69] Pegasus, Pegasus's home page, http://pegasus.isi.edu, ChicagoSim, Last date accessed: January 2006.

[70] Ramamritham, J. A. Stankovic, and W. Zhao, "Distributed Scheduling of Tasks with Deadlines and Resource Requirements", IEEE Transactions On Computers, Vol. 38, No. 8, August 1989.

[71] Ranganathan, ChicSim (The Chicago Grid Computing Simulator), http://people.cs.uchicago.edu/~krangana/ChicSim.html, Last date accessed: January 2005.

[72] Ridge, D. Becker, P. Merkey, and T. Sterling, "Beowolf: Harnessing the Power of Parallelism", in a pile-of-pcs, vol 8, 1994.

[73] SETI@home, http://setiathome.berkeley.edu, Last date accessed: May 2006.

[74] Shin, and Y. Chang, "Load Sharing in Distributed Real-Time Systems with State-Change Broadcasts", IEEE Transactions on Computers, Vol. 38, No. 8, August, 1989.

[75] Shroff, D. Watson, N. Flann, and R. Freund, "Genetic Simulated Annealing for Scheduling Data-dependent Tasks in Heterogeneous Environments", in "5th IEEE Heterogeneous Computing Workshop (HCW '96)," pp. 98-104, 1996.

[76] Siegel and S. Ali, "Techniques for Mapping Tasks to Machines in Heterogeneous Computing Systems", Journal of Systems Architecture, pp. 627-639, 2000.

[77] Silva, R. Buyya, "Parallel Programming Models and Paradigms", High Performance Cluster Computing: Programming and Applications, ISBN 0-13-013785-5, Prentice Hall PTR, NJ, 1999.

[78] Silva, W. Cirne, and F. V. Brasileiro, "Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids", European Conference on Parallel Processing, 2003.

[79] Slowinski and M. Hapke, "Scheduling under Fuzziness", Physica-Verlag, 2000.

[80] Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien, "The MicroGrid: A Scientific Tool for Modeling Computational Grids", Proceedings of IEEE Supercomputing (SC 2000), Nov. 4-10, 2000, Dallas, USA, 2000.

[81] Subramani, R. Kettimuthu, S. Srinivasan, and P. Sadayappan, "Distributed Job Scheduling on Computational Grids using Multiple Simultaneous Requests", Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing HPDC-11 2002 (HPDC'02), IEEE, 2002.

[82] Sulistio and R. Buyya, "A Time Optimization Algorithm for Scheduling Bag-of-Task Applications in Auction-Based Proportional Share Systems", 17th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'05), IEEE, 2005.

[83] SUN Grid Engine, CODINE, http://www.chm.tu-dresden.de/edv/codine, Las date accessed: December 2006.

[84] Szelke and R. M. Kerr, "Knowledge-Based Reactive Scheduling", North-Holland, 1994.

[85] Takefusa, S. Matsuoka, K. Aida, H. Nakada, and U. Nagashima, "Overview of a Performance Evaluation System for Global Computing Scheduling Algorithms", IEEE Computer Society, 1999.

[86] Terwilliger, UCLA Parsec: Programming Language Users Manual, http://pcl.cs.ucla.edu/projects/parsec/manual/#_Toc428871704. Last date accessed: November 2005.

[87] The Globus Alliance, the Globus Toolkit, http://www.globus.org/toolkit, Last date accessed: January 2005.

[88] The GRIDS Labs and the GridBus Project, GridBus, http://www.gridbus.org, last accessed date: November 2006.

[89] UC Berkeley, "Information Technology News Channel", The Regents of the University of California, 2005

[90] Vraalsen, R. A. Aydt, C. L. Mendes, and D.A. Reed, "Performance Contracts: Predicting and Monitoring Grid Application Behavior", in Proceedings of the 2nd International Workshop on Grid Computing (GRID 2001), Lecture Notes in Computer Science (LNCS) Vol. 2242, pp. 154-165, Springer-Verlag, 2001.

[91] Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, "Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach", Parallel Processing Laboratory, Journal of Parallel and Distributed Computing (J. Parallel Distributed Computing), pp. 8-22, 1997.

[92] Wu, W. Shu, and H. Zhang, Segmented Min-Min: "A Static Mapping Algorithm for Meta-tasks on Heterogeneous Computing Systems", IEEE, 2000.

[93] Xiaoshan, X. Sun, and G. von Laszewski, "QoS Guided Min-Min Heuristic for Grid Task Scheduling", Department of Computer Science, Illinois Institute of Technology, IL, USA, May 2003.

[94] Yarkhan and J. Dongarra, "Experiments with Scheduling Using Simulated Annealing in a Grid Environment", In M. Parashar, editor, Lecture notes in computer science 2536 Grid Computing - GRID 2002, volume Third International Workshop, pages 232 242, Baltimore, MD, USA, November 2002.

[95] Young, S. McGough, S. NewHouse, and J. Darlington, "Scheduling Architecture and Algorithms within the ICENI Grid Middleware", In Proc. of UK e-Science All Hands Meeting, pp.512, Nottingham, UK, September 2003.

[96] Yu, D. C. Marinescu, and A. S. Wu, "A Genetic Approach to Planning in Heterogeneous Computing Environments", Parallel and Distributed Processing Symposium, IEEE, 2003.

[97] Zhang, "Scheduling Algorithm for Real-Time Application in Grid Environment", IEEE, 2002.

[98] Zhu, "A Survey on Grid Scheduling Systems", Department of Computer Science, Hong Kong University of Science and Technology, 2005.